ATLAS

Rapport de Projet

LU2IN013 - Simulation d'élections

ATLAS
CHU Amélie
COMBO Leyna
PHAM Anabelle
SAE LIM Thierry

Année 2023-2024
Chargés: Nicolas Maudet

Sommaire:

1. Introduction

Le but de l'UE est de pouvoir mener un projet donné en équipe. L'objectif de ce projet est de mettre en place un simulateur de vote. La motivation est de pouvoir à la fin, simuler une élection et afficher les résultats en fonction des différentes méthodes de vote implémentées.
Tout au long du projet, nos tâches vont évoluer en fonction de la demande du client (nos chargés de TME).

Le projet s'est déroulé sur 3 sprints:

SPRINT 1: Le but de ce sprint est de créer une plateforme où il sera possible de simuler une élection. Il faudra implémenter au moins 3 méthodes de vote, choisir les paramètres de notre élection et visualiser les résultats en fonction de la méthode de vote. Les candidats et les électeurs doivent pouvoir être placés aléatoirement et manuellement sur la grille d'élection.

SPRINT 2: Dans ce second sprint, on doit travailler sur une extension au choix. Nous avons choisi d'implémenter dans notre site le fait de pouvoir élire non plus un seul candidat, mais un comité, ensemble de candidats.

SPRINT 3: Ce sprint de 4 heures avait pour objectif d'implémenter sur notre site la simulation d'une démocratie liquide. Les votants pourront de façon aléatoire, choisir de déléguer leur vote ou non à un autre votant.

2. Manuel utilisateur

Pour lancer le site, il faut tout d'abord télécharger le dossier sur le **gitlab**. Ce dossier contient deux dossiers principaux:

- documentation : dossier contenant la documentation du code
- Site: dossier contenant tous les fichiers permettant de lancer le site

Une fois le dossier ouvert, se rendre dans le dossier **Site** et lancer la commande suivante : **python code_final.py**

```
cd Site
python code_final.py
```

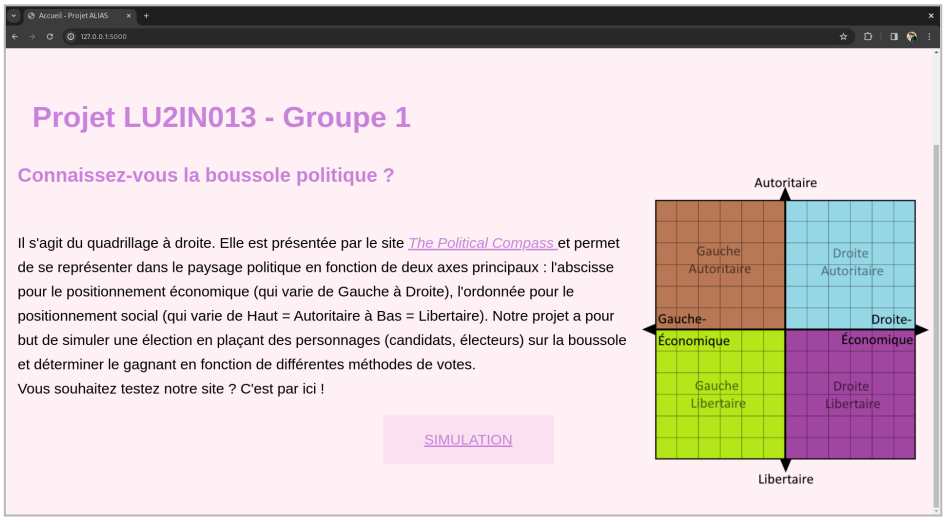
Lancer ensuite le serveur local en appuyant sur l'URL qui apparaît par la suite (ici l'URL généré est `http://127.0.0.1:5000`)

```
 * Serving Flask app 'code_final'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a
production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

La site va se lancer en serveur local sur le navigateur par défaut.

Le site se décrit en plusieurs pages comme suit:

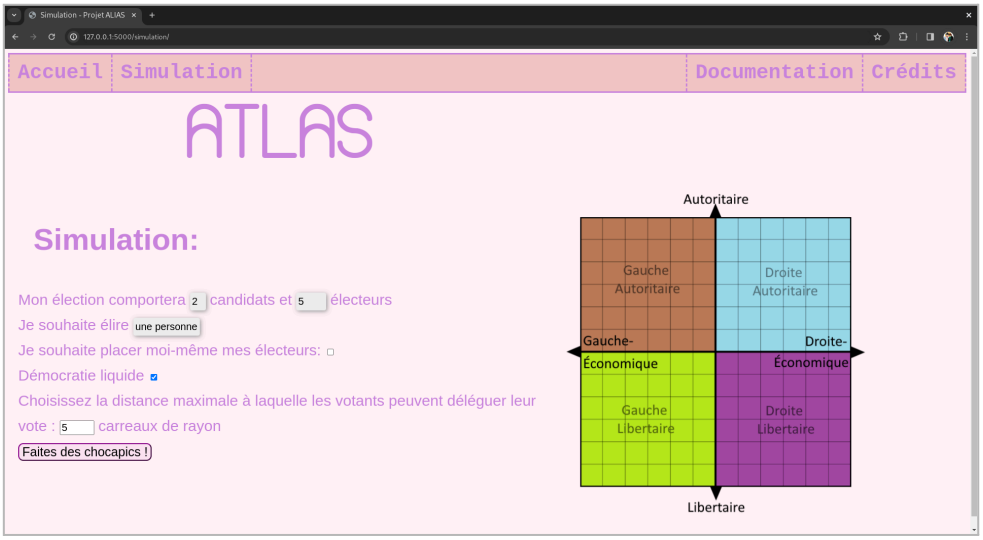
Accueil du site (**site.html**) :



- Elle contient une présentation rapide du projet ainsi qu'un accès à la page de simulation. La barre de navigation permet d'aller vers les pages de Documentation (renseignant les différentes méthodes de vote) et de crédits.

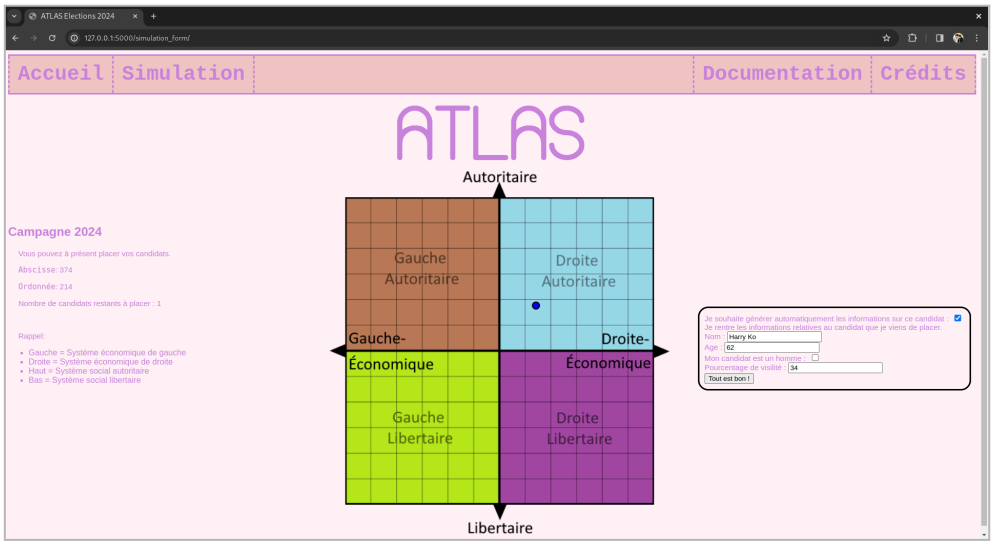
Sur la page **Simulation**, il est demandé de sélectionner des données et des paramètres de la simulation :

- Choix principaux (**simulation.html**) :



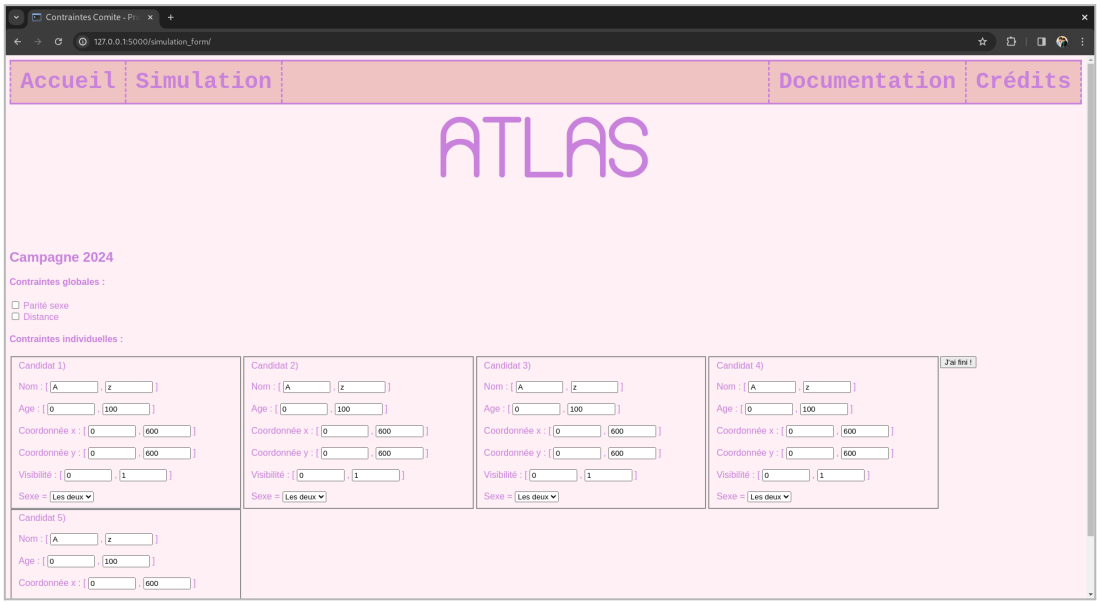
Nombre d'électeurs et de candidats, type d'élection (individuelle / collective), type de placement (manuel / automatique), adoption de la démocratie liquide

- Entrée des données manuelles :



L'utilisateur place tous les candidats avant de placer tous les électeurs. À chaque candidat placé, une boîte de dialogue s'ouvre pour entrer ses informations (nom, âge, sexe, visibilité). L'utilisateur a également la possibilité de laisser ces informations être remplies automatiquement. (**placement_m.html**)

- Entrée des contraintes globales et individuelles pour l'élection d'un comité (**comite.html**)



Sur cette page il est possible de choisir:

- l'ajout ou non d'une contrainte globale sur la parité du sexe
- l'ajout ou non d'une contrainte globale sur la distance entre chacun des candidats élus

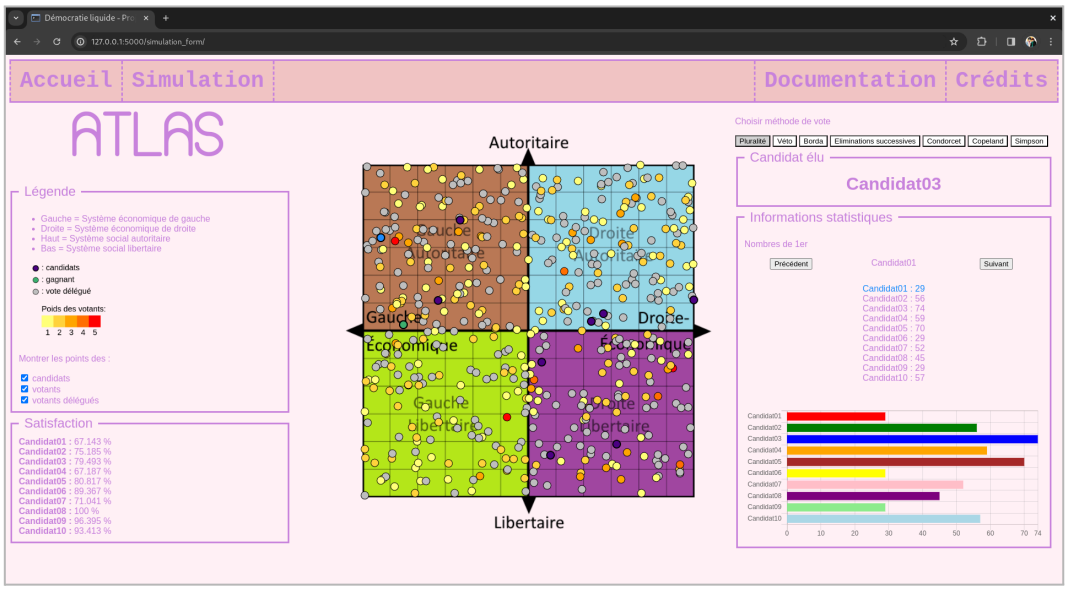
- la formulation des contraintes individuelles sur le nom, l'âge, les coordonnées x et y, la visibilité, et le sexe du candidat

Les pages d'affichage des résultats :

- Individuelle (**affichage_election.html**) :

- Affiche la boussole de l'élection en fonction des données.
- Choix de l'affichage en fonction de la méthode de vote choisie.
- Les statistiques des candidats en fonction de la méthode de vote choisie (graphe, etc) sont également affichées.

- Individuelle liquide (**liquide.html**) :



- Affiche la boussole de l'élection en fonction des données.
- Choix de l'affichage en fonction de la méthode de vote choisie.
- Les statistiques des candidats en fonction de la méthode de vote choisie (graphe, etc) sont également affichées.
- Les électeurs ayant délégué leur vote sont grisés et peuvent être cachés.

- Comité (**affiche_comite.html**) :

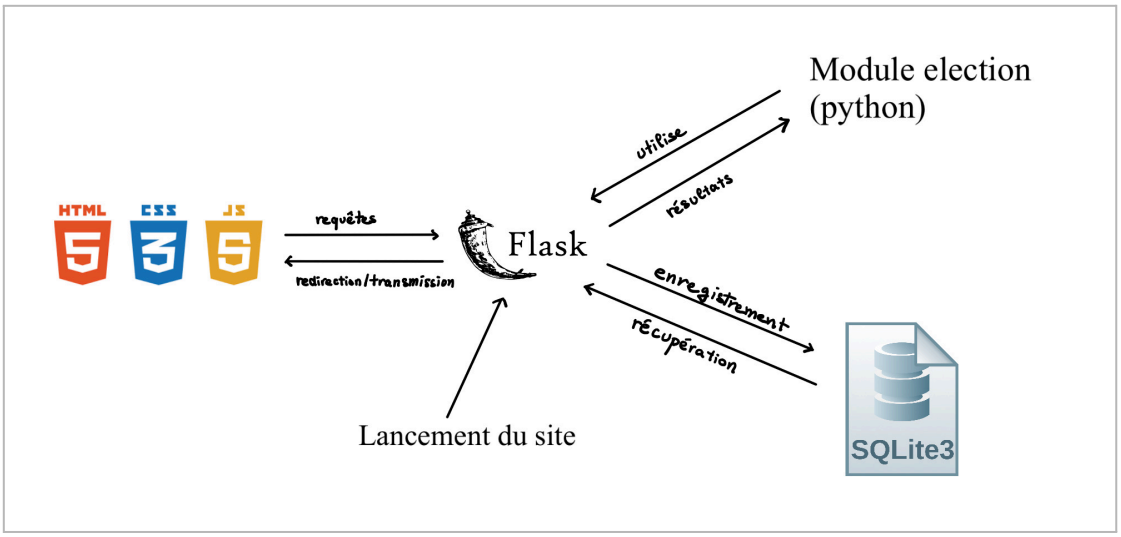
- Affiche la boussole de l'élection en fonction des données.
- Choix de l'affichage en fonction de la méthode de vote.
- Affichage (tooltip) des attributs de chacun des candidats.
- Affichage des contraintes sur l'élection du comité.
- Affichage de la liste des candidats.

3. Architecture du projet

Pour notre projet, nous avons choisi de partir avec Python, étant un langage malléable et facile d'utilisation et le plus maîtrisé, et HTML pour l'interface utilisateur, accompagné de JS et de CSS.

Afin de combiner ces langages, nous avons décidé de travailler avec Flask, qui est un outil que nous avons déjà utilisé auparavant, permettant de lancer un site web sous Python. Les langages utilisés sont ainsi: **HTML, CSS, JS et Python**.

Schéma de l'architecture de notre site



Notre site fonctionne donc sur un serveur local qui est lancé en python grâce à **Flask**. Ce dernier est un module qui permet de créer une application web en HTML et de l'ordonner avec python. Il est de même possible d'y manipuler SQLite3, une bibliothèque qui permet d'utiliser une base de données en SQL.

Grâce à ce module et au lien avec HTML, nous avions le champ libre sur l'interface de notre site, ainsi que la possibilité d'utiliser une base de données pour sauvegarder les coordonnées et attributs de nos candidats et de nos votants. L'usage d'une BDD évite de surcharger le cache du serveur en sauvegardant sur la session les données de notre élection et de ce fait il est possible de recharger les pages sans perdre nos résultats.

4. Rapport de test

Il y a deux fichiers tests : **testR.py** et **testS.py**. Les deux fichiers recensent respectivement la totalité des tests avec assertions utilisées pour l'agrégation du **Ranking** et l'agrégation du **Scoring**.
Chaque fichier est séparé par sections de tests, où une section de test est représentée par une fonction qui teste une fonctionnalité particulière. Chaque section est composée de plusieurs assertions qui testent des fonctions mineures utilisées pour la fonctionnalité de la section.

Concernant les limites de calculs, le nombre de candidats choisi sera majoré par **25**, pour cause d'utilisation de la combinatoire lors de l'élection d'un comité. Hormis l'élection de comité, le nombre de candidats passé en paramètre des fonctions de vote peut atteindre de grandes valeurs. Le nombre de votants quant à lui, peut atteindre facilement les milliers, quelle que soit la situation.

Ces tests des limites de calculs ne possèdent aucune trace (elles ne sont pas écrites dans un fichier) puisqu'elles dépendent du support utilisé. L'ordinateur utilisé lors des tests était un support plus puissant que la moyenne, les limites de calculs sont donc supérieures à ce qu'un support normal peut expérimenter. Le site majorera donc le nombre de candidats à 20 (quel que soit le type de vote).

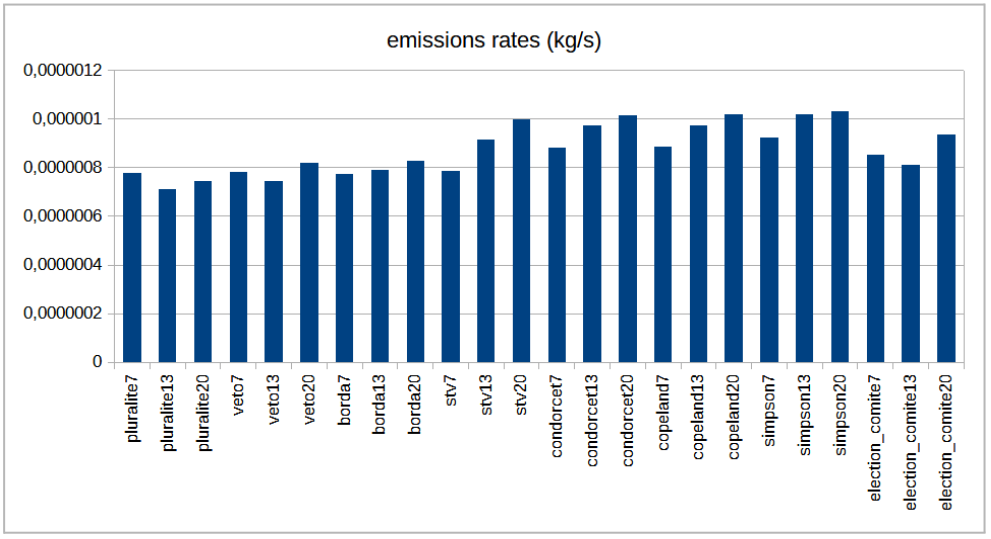
5. Rapport carbone

Nous avons évalué le rapport carbone de notre logiciel à l'aide du module **Codecarbon**. Cette évaluation se déroule en deux temps: nous mesurons tout d'abord le rapport carbone propre à chacune de nos fonctions simulant une méthode de vote, en faisant varier le nombre de candidats et le nombre d'électeurs, ensuite nous réalisons un rapport carbone sur l'ensemble du site.

La mesure se fera sur 3 paramètres: la durée de l'exécution de la fonction (en secondes), l'émission [CO_2eq] en Kg et en Kg/s.

Chaque fonction est appelée avec des paramètres variants, afin de voir l'impact de l'émission carbone en fonction de ces variables. Chaque nom de fonction est suivi du nombre de candidats passé en paramètre.

Courbes associées aux tests



Pour chaque méthode de vote que nous disposons, nous les avons lancés avec pour paramètre 7, puis 13, puis 20 candidats afin d'observer l'impact du nombre de candidats sur l'émission carbone. Comme prévu, l'augmentation du nombre de candidats augmente l'émission de carbone, à certaines exceptions qui sont expliquées par l'aléa.

Vient ensuite la mesure du site dans l'intégralité. La durée n'est ici pas prise en compte étant donné que le temps passé sur le site dépend de l'utilisateur et n'influence pas réellement les données que l'on souhaite récolter (le site une fois lancé consomme lui-même une certaine quantité d'énergie).

Ce qui nous intéresse ici c'est l'émission d'énergie émis par le site pour un type de vote en particulier et pour un nombre de candidats différent.
Pour les mesures du site, 7 simulations ont été réalisées:

- Élection d'un **comité** de 2 candidats pour 5 candidats et 20 votants en **automatique**
- Élection d'un **comité** de 5 candidats pour 12 candidats et 50 votants en **automatique**
- Élection d'un **comité** de 2 candidats pour 5 candidats et 20 votants **manuelle**
- Élection d'un **candidat** avec 5 candidats et 20 votants en **automatique**
- Élection d'un **candidat** avec 15 candidats et 100 votants en **automatique**
- Élection d'un candidat avec 5 candidats et 20 votants en **manuelle**
- Élection **démocratie liquide** d'un **candidat** avec 15 candidats 50 votants en **automatique**

On constate au niveau des données, que la moyenne de ces 7 opérations donne une émission de 1,05184 mg/s. Nous avons constaté qu'entre chaque opérations, la différence d'émissions de carbone en kg/s est très minime et très influencée par l'aléa, d'où le choix de faire la moyenne de ces opérations.