## Conception

## Cas d’utilisation (Diagramme disponible ici)

## Initialiser la simulation

Ici, l'utilisateur aura le choix entre plusieurs méthodes d'initialisation. Premièrement il pourra initialiser sa simulation avec un jeu de données préchargées dans un fichier texte. Deuxièmement il pourra également générer des données aléatoires le temps accessible depuis une fonctionnalité de l'interface.

## Gérer ses données

À la condition d'avoir initialisé sa simulation, l'utilisateur peut également gérer ses données en les sauvegardant vers un fichier ou en les chargeant depuis un fichier.

## Lancer la simulation.

Le lancement de la simulation est l'action déterminant le résultat des paramètres fournis par celle-ci. Cette étape regarde avoir initialisé celle-ci avec un jeu de données valide, sélectionnez un mode de scrutin. Son fonctionnement peut également être étendu par l'évolution des opinions des électeurs. Cela peut être réalisé de 2 manières différentes soit par sondage d'une portion des électeurs soit par interaction socio-politique.

## Classe (Diagramme disponible ici)

## Personne

Les 2 catégories politiques : électeur et candidat, sont représentés par 2 classes distinctes candidat héritant d’électeurs. Elle possède respectivement 2 axes, pouvoir d'achat et écologie. Cela nous permettra de produire un espace à 2 dimensions dans lequel nous ferons nos opérations de comparaison d'influence vis-à-vis des autres électeurs et candidats.

## SaveManager

Ici un outil de gestion de sauvegarde a été produit nommer SaveManager, il est utilisé pour charger où sauvegarder toute collection élément implémentant l'interface CSVReady. Il est notamment utilisé pour charger où sauvegarder des listes d'électeurs ou de candidats.

## Interaction dynamique

Nous avons choisi d’abstraire l'ensemble des interactions dynamiques en une interface : interactions dynamiques. Les interactions dynamiques par sondage ou par en relation socio-politique implémente cette dernière. La première réalisa sondage d'une fraction des électeurs sur l'ensemble des candidats, on influence ensuite l'ensemble des électeurs par le ainsi obtenu des candidats cette influence est multiple, elle peut être simple (les électeurs se déplacent d'une distance fixe vers le candidat qui l'intéresse le plus), par utilité (l'intérêt que porte un électeur pour un candidat est constitué de l'inverse de la distance qui les sépare par le nombre de voix obtenues lors du sondage) et enfin le dernier mode «  utilité multiple » ou le déplacement de l'électeur vers son candidat favori est proportionnelle à l'utilité calculée comme précédemment. Nous nous sommes grandement inspiré du modèle Iznogoud.

## Scrutin

5 scrutins ont été demandés 5 scrutins ont été implémentés de la manière suivante. Ceux-ci implémentent la classe abstraite « Scrutin » contenant une méthode «GetClassementCandidat» (calcul le classement des candidats) ainsi qu'une méthode « vérifietableau » Vérifiant la mise en forme des jeux de données électeurs et candidats passer en paramètre du constructeur.

Nous avons procédé de la manière suivante pour implémenter les scrutins :

Le scrutin majoritaire 2 tours est composé de 2 scrutins majoritaires un tour. Le scrutin alternatif a été réalisé par récurrence.

## Extensions

## Environnement de travail

Afin de mieux nous organiser, nous avons choisi d'utiliser le gestionnaire de versions git. Notre repo se trouve à l'adresse suivante : <https://github.com/Les-Techos/Election-APO> . Plusieurs branches ont été utilisées au cours de notre développement ainsi qu'une technique de mer je appeler « squash and merge » compact ainsi les commits faits sur les autres branches en un seul sur la branche main.

## Installation

en ce qui concerne la construction du projet, nous avons choisi d'utiliser le builder Gradle. Celui-ci est disponible en version standalone offrant la possibilité à quiconque téléchargeant le projet de construire et de lancer le programme à partir d'une connexion internet. ce dernier permet également de générer la javadoc ainsi que de réaliser les tests unitaires fournis par Junit. L’ensemble des commandes utiles sont décrites dans le README.

## Tests unitaires

afin de vérifier que les modifications apportées à chaque étape du projet n'entrave pas le bon fonctionnement initial du programme, nous avons créé une série de tests exécuter par moment de la construction. Ces tests se trouvent au chemin suivant :

[Election-APO](https://github.com/Les-Techos/Election-APO)/[app](https://github.com/Les-Techos/Election-APO/tree/main/app)/[src](https://github.com/Les-Techos/Election-APO/tree/main/app/src)/[test](https://github.com/Les-Techos/Election-APO/tree/main/app/src/test)/[java](https://github.com/Les-Techos/Election-APO/tree/main/app/src/test/java)/main/AppTest.java

## Sauvegarde

Héritant de CSVReady, les candidats et électeurs bénéficient d'une méthode « toCSVString » utilisée SaveManager pour sauvegarder/charger ces dernières vers/depuis un fichier. Il nous est possible de sauvegarder toute collection (Hashet, Map, ArrayList etc…) Dont le résultat d'une élection qui est une ArrayList de candidats ainsi que les électeurs influencés étant un HashSet D’électeurs.

## Interface graphique

/Fares

## Spatialisation

Cette partie a été réalisée en ajoutant un identifiant social pour chaque électeur. celui-ci est modélisé à l'aide d'un entier. il est utilisé à des fins de comparaison lors des interactions socio-politique. cela permet de segmenter la population des électeurs en cercles sociaux autorisant les interactions qu’en leur sein.

## Annexe

## Diagramme de classes

Figure 1 - Diagramme de classe simplifié

## Diagramme de cas d’utilisation