

Cahier des Charges

Vote Itéré de Comité

Par les étudiants

Loic Urien, Victor Racine, Yasmine Hamdani

pour

M. Nicolas Maudet et M. Paolo Vappiani



Projet-ANDROIDE

Master informatique, parcours ANDROIDE

Faculté d'ingénierie - Sorbonne Université

Jeudi 1 Mars 2018

Version du document	0	Date du document	11/02/2018
Auteur(s)	Loïc URIEN Victor RACINE Yasmine HAMDANI	Type de diffusion	Document électronique (pdf)
Maîtres d'ouvrage	Nicolas MAUDET Paolo VAPPIANI		
Date / Signature		Date / Signature	

SOMMAIRE

1. Introduction	2
1.1. Contexte	2
1.2. Description	3
2. Description de la demande	3
2.1) Demande textuelle	3
2.2) Fonctionnalités obligatoires	4
2.2.a) Ecriture et chargement d'un fichier de configuration de simulation	4
2.2.b) Stratégie basée sur la distance de Hamming et agents omniscients et k-approval	4
2.2.c) Représentation graphique de l'avancement de la simulation	4
2.2.d) Export des résultats	5
2.2.e) Reconnaissance des cycles et arrêt automatique	5
2.2.f) Simulation pas à pas, définition de la taille du pas de temps et retour en arrière	5
2.2.g) Génération aléatoire des préférences des agents	6
2.3) Fonctionnalités optionnelles	6
2.3.a) Implémentation de multiples stratégies, niveaux de connaissances, règles de votes et types de préférences	6
2.3.b) Représentations graphiques alternatives	7
2.3.c) Recharger une simulation à un état donné	7
2.4) Fonctionnalités imaginées	7
2.4.a) Changement dynamique de stratégies sous conditions	7
2.4.b) Statistiques d'évolution de la simulation	7
2.4.c) Taille de comité automatique	8
2.4.d) Amélioration de la génération des préférences des agents	8
2.5) Expression fonctionnelle des besoins	8
2.5.a) Exécution de la simulation	8
2.5.b) Sauvegarde des résultats	9
3. Maquettes graphiques	10
3.1) Fenêtre principale	10

3.1.1) Représentation graphique par défaut	12
3.2) Fenêtre de configuration du profil	14
3.3) Barre des menus	16

1. Introduction

1.1. Contexte

Ce projet s'inscrit dans le cadre de l'UE P-ANDROIDE de 1ère année du master informatique parcours ANDROIDE de la faculté d'ingénierie de Sorbonne Université (du 15/01/2018 au 7/01/2018). Il est réalisé par une équipe composée de deux encadreurs enseignants-chercheurs au laboratoire d'informatique de Paris 6 (Lip6) ainsi que de trois étudiants de l'université.

Ce document représente le cahier des charges du projet "Vote itéré de comité" et a pour but d'explicitier et de regrouper la demande des clients, M. Nicolas Maudet et M. Paolo Vappiani, et l'offre du prestataire, le groupe du projet dont les membres sont Loic URIEN, Victor RACINE et Yasmine HAMDANI. Il servira de référence lors du développement du projet et permettra aux deux parties d'être en accord sur les différents points conceptuels avant de se lancer dans le développement de l'application.

Les clients, M. Maudet et M. Vappiani, proposent de créer une application permettant de simuler un vote itéré de comité, en laissant libre recours au groupe sur le choix de certaines fonctionnalités tout en définissant les fonctionnalités obligatoires, sans lesquelles le projet ne pourra être considéré comme complété. Ce document est donc un moyen de synthétiser la demande des clients, de définir les contraintes liées au développement ainsi qu'à l'utilisation du produit fini et d'établir le planning prévu pour réaliser ce projet.

1.2. Description

Le programme, relevant du domaine de la prise de décisions collective, devra permettre la simulation d'un vote itéré de comité. Ceci nécessitera la mise en œuvre de notions de décision dans un cadre multi-agents dynamique.

L'intérêt de cette application est de permettre l'étude de l'évolution d'un vote itéré effectué par un certain nombre d'agents (électeurs) ayant à priori des préférences électorales différentes, l'objectif étant d'arriver à un consensus entre les agents afin d'élire les membres d'un comité. Dans la suite du document, nous référerons à chaque itération du vote comme étant un "pas de temps".

Le programme devra offrir une interface simple permettant d'ajuster les paramètres de la simulation (notamment les préférences des votants, leur niveau d'information, et le nombre maximum d'itérations) et afficher le résultat du vote.

L'application se voudra modulaire afin de faciliter l'ajout/suppression et la modification de ses composants, en particulier les stratégies, les degrés de connaissances associés et les représentations graphiques de la simulation.

2. Description de la demande

2.1) Demande textuelle

L'étude efficace des différents comportements d'un groupe votant pour l'élection d'un comité représente un intérêt certain dans notre société actuelle, aussi bien dans le domaine de la politique que pour l'organisation et la planification. Dans cette optique, nous nous proposons de mettre en place un simulateur de ce type de comportements, capable de modéliser diverses stratégies et préférences.

Le programme ainsi créé devra permettre à l'utilisateur de configurer sa simulation via une interface simplifiée, ainsi que de la sauvegarder afin de permettre une réutilisation ou modification de celle-ci à une date ultérieure. L'application devra également permettre une visualisation simple des résultats de la simulation, plusieurs options d'avancement de la simulation (pause, pas à pas, avance rapide, etc ...) ainsi que la possibilité d'exporter un rapport détaillé sous forme de document CSV à chaque pas de temps afin de pouvoir étudier les résultats.

2.2) Fonctionnalités obligatoires

2.2.a) Ecriture et chargement d'un fichier de configuration de simulation

Afin de faciliter l'utilisation du simulateur, il est de la plus haute importance de proposer une manière simple de définir la simulation ainsi que de pouvoir y revenir plus tard. Dans ce cadre, nous nous proposons de donner à l'utilisateur la possibilité de sauvegarder sa simulation dans un fichier JSON, afin de pouvoir la charger plus tard à nouveau dans le simulateur.

Cette configuration se devra de sauvegarder les agents, leurs préférences, les candidats ainsi que le type de préférence utilisée et la stratégie mise en application. Elle ne gardera en aucun cas quelconque mémoire de la simulation à un temps donné.

2.2.b) Stratégie basée sur la distance de Hamming et agents omniscients et k-approval

L'application devra implémenter au minimum une stratégie basée sur la distance de Hamming en utilisant des agents ayant une connaissance totale de leur environnement, à savoir, une connaissance du nombre de votes reçus par chaque candidat après chaque pas de temps. Le comité sera élu par la règle de vote "k-approval", à savoir les k candidats ayant reçu le plus de votes. S'il y a égalité, nous opterons pour un choix par ordre lexicographique.

2.2.c) Représentation graphique de l'avancement de la simulation

Une représentation graphique dynamique de la simulation devra être mise en place afin de permettre à l'utilisateur de visualiser la progression ou l'absence de progression, lui permettant d'arrêter manuellement la simulation si besoin.

Par défaut, la représentation sera un simple graphe ayant sur l'axe des abscisses les différents candidats et en ordonnées leurs scores. Pour plus d'informations, voir la section 3.1.1.

(TODO : décrire brièvement les représentations attendues, il faudra en parler à Maudet afin d'avoir une idée plus claire)

2.2.d) Export des résultats

L'utilisateur devra pouvoir être en mesure d'exporter le résultat de la simulation à chaque pas de temps sous forme d'un fichier CSV contenant les votants avec leurs préférences actuelles ainsi que le résultat des votes. Il n'est cependant pas prévu de pouvoir charger ce fichier dans l'application afin de reprendre à un état donné de la simulation (voir : Fonctionnalités optionnelles)

2.2.e) Reconnaissance des cycles et arrêt automatique

Nous devons être en mesure de détecter les cycles et effectuer les actions nécessaires à la non stagnation de la simulation. Dans les fonctionnalités obligatoires, nous arrêterons la simulation immédiatement après avoir détecté un cycle ou une stagnation de durée N, indiquant l'arrivée probable à un état de stabilité, avec N un nombre de pas de temps configurable. L'application devra donc garder en mémoire un certain nombre d'états précédents de la simulation afin de pouvoir détecter les cycles efficacement.

2.2.f) Simulation pas à pas, définition de la taille du pas de temps et retour en arrière

Il est nécessaires de pouvoir mettre en pause la simulation à n'importe quel instant, la redémarrer ou l'avancer pas à pas. Il est également nécessaire de pouvoir régler la

vitesse de la simulation, l'utilisateur devra donc être en mesure de définir la taille d'un pas de temps manuellement. L'application devra de plus garder en mémoire un certain nombre d'états précédents afin de permettre le retour en arrière de la simulation. De plus, à chaque étape, le comité élu devra être annoncé à l'utilisateur.

2.2.g) Génération aléatoire des préférences des agents

L'application devra pouvoir générer de manière aléatoire les préférences des agents étant donné une liste de candidats. Dans un premier temps, nous nous concentrerons uniquement sur une génération purement aléatoire et uniforme.

2.3) Fonctionnalités optionnelles

2.3.a) Implémentation de multiples stratégies, niveaux de connaissances, règles de votes et types de préférences

Il serait extrêmement intéressant pour la simulation d'avoir la possibilité de changer la stratégie utilisée par les agents, afin de ne pas se limiter uniquement à la distance de Hamming. Il serait aussi préférable d'implémenter d'autres niveaux de connaissances que celui d'omniscience (i.e : les agents connaissent à tout moment le résultat des votes). Nous pouvons noter les alternatives suivantes en ce qui concerne la stratégie des agents :

- Préférence responsive
- ... (TODO)

Toutes les stratégies ne peuvent être utilisées avec un niveau de connaissance quelconque, il est donc évident que le niveau de connaissance dépendra grandement de la stratégie utilisée. Il est cependant possible d'avoir plusieurs niveaux de connaissances pour une stratégie donnée.

2.3.b) Représentations graphiques alternatives

Nous pouvons envisager d'adopter d'autres modes de représentations graphiques que celui par défaut, afin de donner d'autres visions de la simulation à l'utilisateur. Nous pouvons également envisager un layout interactif permettant de visualiser plusieurs paramètres en même temps, donnant ainsi plus de précisions sur l'évolution du modèle.

2.3.c) Recharger une simulation à un état donné

Bien que non prévu dans les fonctionnalités de base de l'application, il serait intéressant de penser à un moyen de recharger une simulation à un instant donné. L'application pourra ainsi recharger un fichier de résultat CSV ainsi que la configuration correspondante (identification unique par code de hachage) afin de reprendre la simulation là où elle s'était arrêtée. Il est également possible de repenser le modèle d'export des résultats afin que la configuration soit incluse, évitant ainsi de ne plus pouvoir réutiliser le fichier de résultats si le fichier de configuration n'est plus présent.

2.4) Fonctionnalités imaginées

2.4.a) Changement dynamique de stratégies sous conditions

Il serait intéressant de pouvoir effectuer un ou plusieurs changements de stratégie si une ou plusieurs conditions sont remplies. On peut imaginer par exemple qu'à partir d'un certain nombre d'itérations, ou après avoir détecté un cycle ou un état de stabilité avec un trop grand désaccord entre les agents, l'application change de stratégie avec d'essayer de trouver une meilleure solution. Les conditions pourraient être entrées par l'utilisateur ou bien simplement prédéfinies par l'application.

2.4.b) Statistiques d'évolution de la simulation

L'application pourrait donner des statistiques sur la vitesse de convergence et la distribution des votes.

2.4.c) Taille de comité automatique

Nous voulons être en mesure de pouvoir déterminer automatiquement une bonne taille de comité sans que l'utilisateur n'ait besoin de la définir, en utilisant par exemple une règle de type min-max. Précisons cependant que l'utilisateur devra être en mesure de remplacer cette valeur automatique par une valeur de son choix afin de ne pas limiter les possibilités de simulation.

2.4.d) Amélioration de la génération des préférences des agents

Il serait intéressant d'utiliser une génération plus "intelligente" et paramétrable pour la génération des préférences des agents, par exemple en utilisant la méthode de Packett-Luce.

2.5) Expression fonctionnelle des besoins

2.5.a) Exécution de la simulation

L'application devra pouvoir effectuer la simulation jusqu'à rencontrer une condition d'arrêt ou que l'utilisateur décide de lui-même d'y mettre fin. L'interface devra permettre de facilement contrôler la taille du pas de temps ainsi que l'avancée de la simulation (TODO : maquette d'exemple). L'utilisateur devra pouvoir :

- Mettre la simulation en pause
- Relancer la simulation en course
- Remettre la simulation à l'état initial
- Effectuer une avancée pas à pas (uniquement disponible en mode pause)
- Arrêter la simulation
- Revenir en arrière

A chaque pas de temps, sauf au pas de temps initial, l'utilisateur aura la possibilité d'enregistrer les résultats obtenus dans un fichier CSV. (voir 2.5.b)

L'échec lors du chargement d'une configuration à partir d'un fichier devra résulter en une simulation vide et un message d'erreur explicite devra être fourni en log. L'application ne devra en aucun cas produire une simulation erronée à partir d'un fichier de configuration invalide, toute donnée lue devra être ignorée.

Une erreur lors de la simulation devra être traitée comme fatale et les résultats obtenus invalides. L'utilisateur pourra choisir de sauvegarder les résultats obtenus à l'itération actuelle ou durant une itération précédente, mais le programme n'offrira alors aucune garantie de validité sur ceux-ci. De même, lors d'un échec il ne sera plus possible de relancer la simulation, de revenir en arrière ou de faire du pas à pas.

De plus, si la simulation ne peut pas effectuer une étape dans le pas de temps défini par l'utilisateur, l'application devra fournir un message clair concernant cette situation.

2.5.b) Sauvegarde des résultats

A chaque pas de temps, les résultats de la simulation (comité élu, score des candidats) devront pouvoir être sauvegardés dans un fichier au format CSV, pouvant être aisément chargé dans un logiciel de type tableur. Comme indiqué dans la section précédente, si l'application lance une erreur, les résultats ne pourront plus être considérés comme valides. L'utilisateur aura trois moyens de sauvegarder l'itération actuelle, soit par le raccourci ctrl+s, soit le bouton de sauvegarde dans la barre des contrôles (voir section 3.2) soit finalement par le menu "file" en cliquant sur "save results as" (voir section 3.3).

Par défaut, le nom du fichier sera de la forme \$nomDuProfil\$_\$nombreIterationActuel\$, où "nomDuProfil" et "nombreIterationActuel" sont des variables de la simulation.

L'utilisateur ne devra pas avoir la possibilité de sauvegarder les résultats de la simulation à l'instant initial, celle-ci n'ayant pas commencé et les résultats étant donc vides. L'application devra lui signaler cette impossibilité par un message expliquant cet état de faits.

Si une erreur se produit lors de la sauvegarde des résultats, alors aucun fichier ne devra être créé et l'utilisateur devra être informé de l'erreur. Si l'application a rencontré une erreur provenant d'une autre partie de l'application que celle de l'écriture du fichier, alors il faudra tout de même sauvegarder les résultats, en informant l'utilisateur que ceux-ci ont de grandes chances d'être invalides.

2.5.c) Création et enregistrement d'un profil de simulation

L'application devra permettre à l'utilisateur de facilement créer, éditer, sauvegarder et recharger un profil de simulation, afin de permettre une plus grande aisance durant le paramétrage de la simulation. Pour créer et éditer un profil, il aura accès à un outil dédié intégré au programme via une interface graphique (voir section 3.2). Si jamais la configuration de l'utilisateur se trouve être erronée, elle ne devra en aucun cas pouvoir être sauvegardée.

L'utilisateur aura également comme possibilité de charger un profil de configuration déjà existant, afin de l'éditer ou de relancer une simulation antérieure. Si le fichier donné en entrée n'est pas un fichier de configuration de profil valide, l'application ne devra en aucun cas entrer dans un stade d'erreur total et ne devra changer aucune valeur que l'utilisateur aurait pu entrer précédemment, agissant comme si aucune action n'avait été effectuée.

3. Maquettes graphiques

3.1) Fenêtre principale

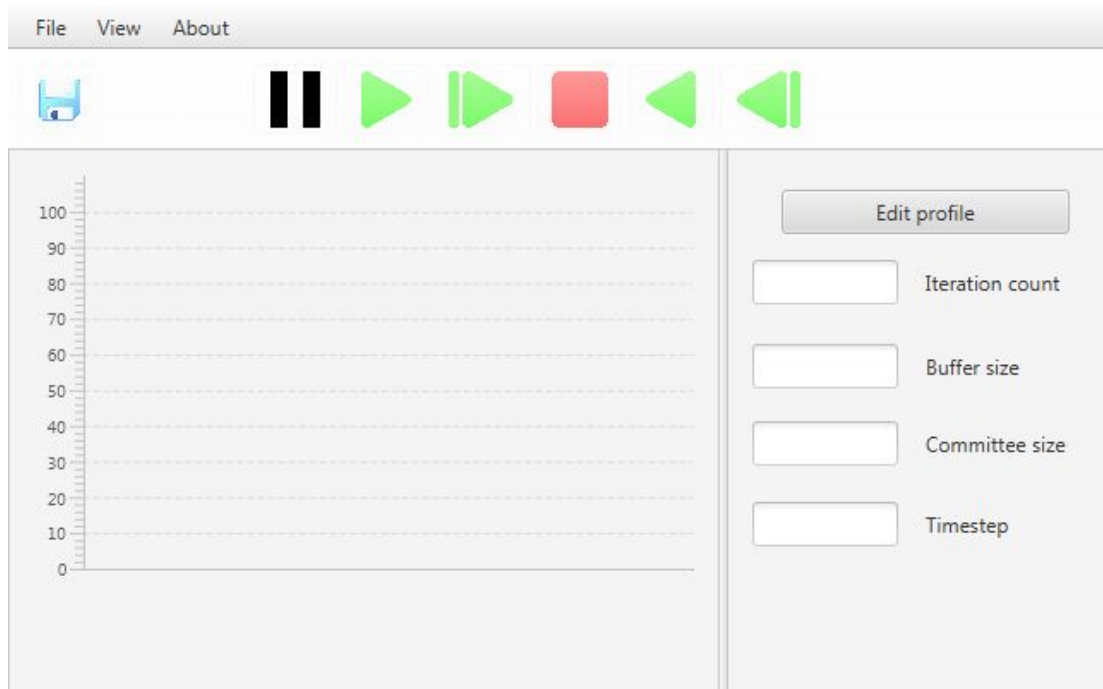


fig 1 : fenêtre principale de l'application

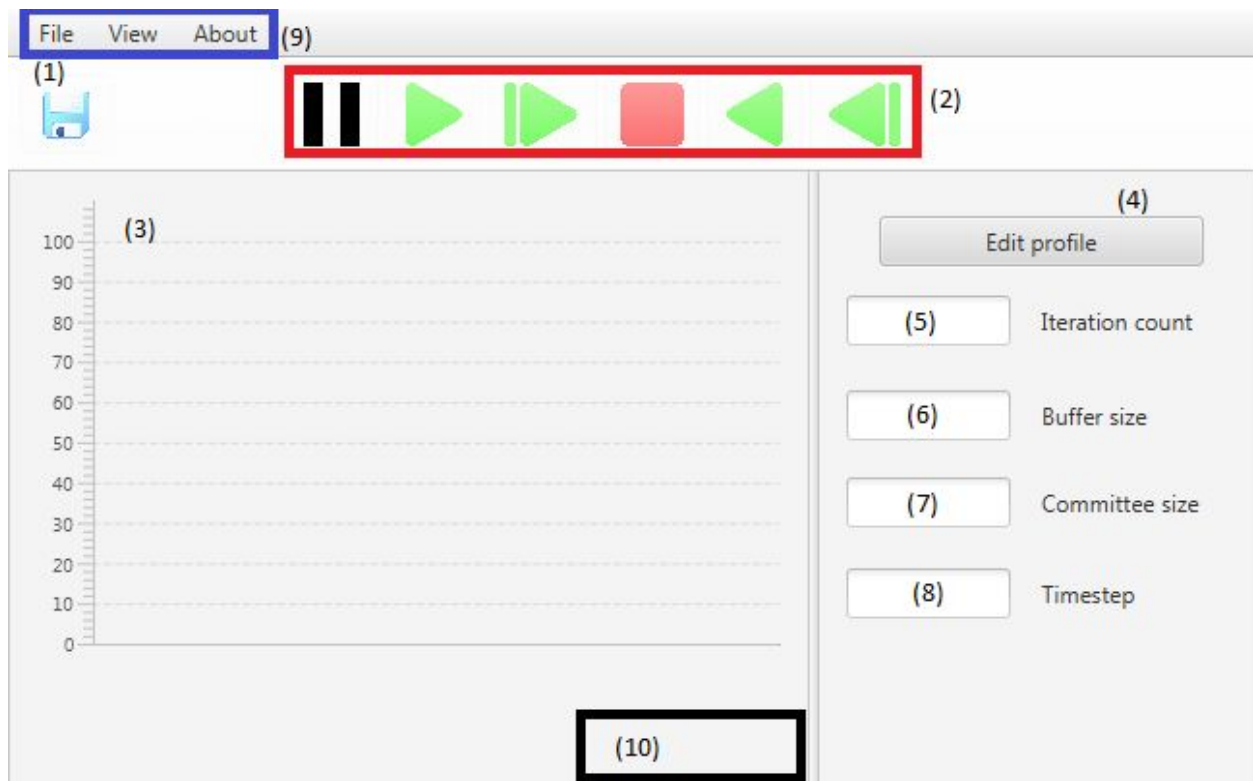


fig 2 : fenêtre principale de l'application annotée

Nous décrivons ici les différentes parties composant l'interface principale de l'application :

- (1) : Sauvegarde la trame actuelle de la simulation au format CSV. Pour plus d'information, se référer à la section 2.5.2
- (2) : Panneau de contrôle de l'avancement de la simulation. De gauche à droite :
 - (a) Met la simulation en pause. Cette option ne doit pas être disponible (bouton désactivé) si jamais l'application est arrêtée
 - (b) Lance la simulation, ou la relance si celle-ci est en pause
 - (c) Effectue une avancée pas à pas de la simulation
 - (d) Arrête la simulation
 - (e) Fait revenir la simulation sur ses pas
 - (f) Retour en arrière pas à pas
- (3) : Affichage des résultats de manière graphique
- (4) : Edition du profil. Ouvre la fenêtre d'édition du profil de la simulation (voir section suivante)
- (5) : Nombre d'itérations avant que la simulation ne s'arrête, infini si laissé vide

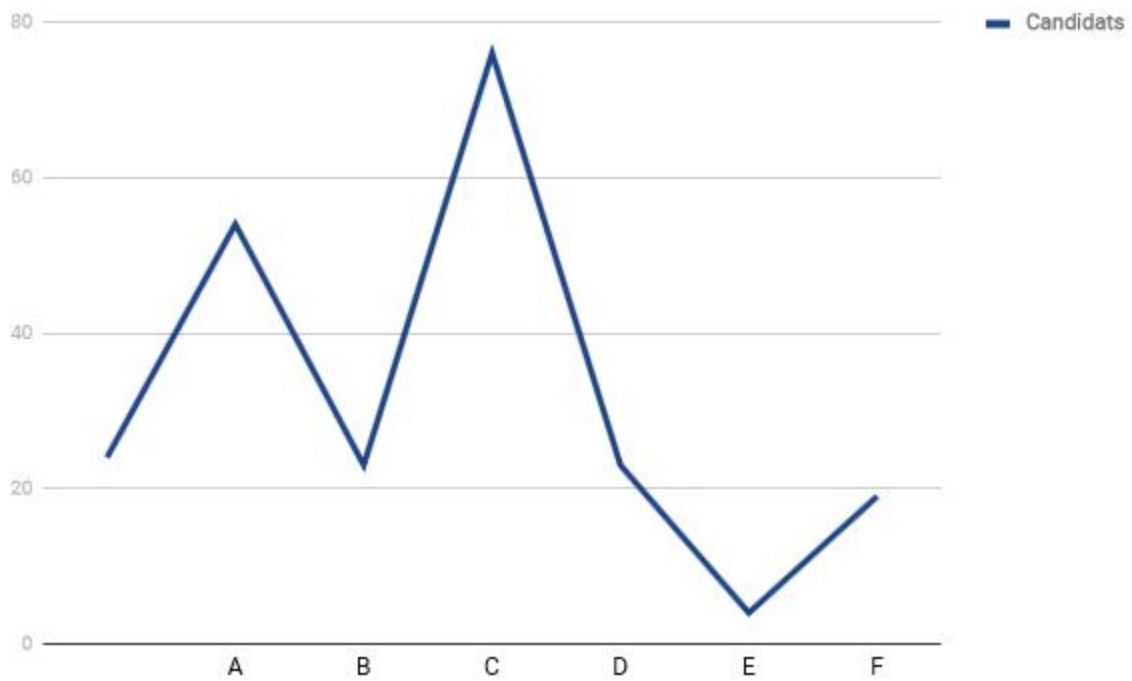
- (6) : Taille de la mémoire tampon de l'application, définissant le nombre de "retours en arrière" pouvant être effectués. Le choix de la valeur par défaut sera laissée à la discrétion des développeurs
- (7) : Taille du comité désirée, comité unitaire par défaut
- (8) : Taille du pas de temps. Si ce champ est laissé vide, l'application essayera de compléter chaque itération le plus rapidement possible, le pas de temps ne sera donc pas constant
- (9) : Barre de menus. Pour plus d'information, voir la section 3.3
- (10) : Indication du comité élu actuellement, vide à l'étape initiale

3.1.1) Représentation graphique par défaut

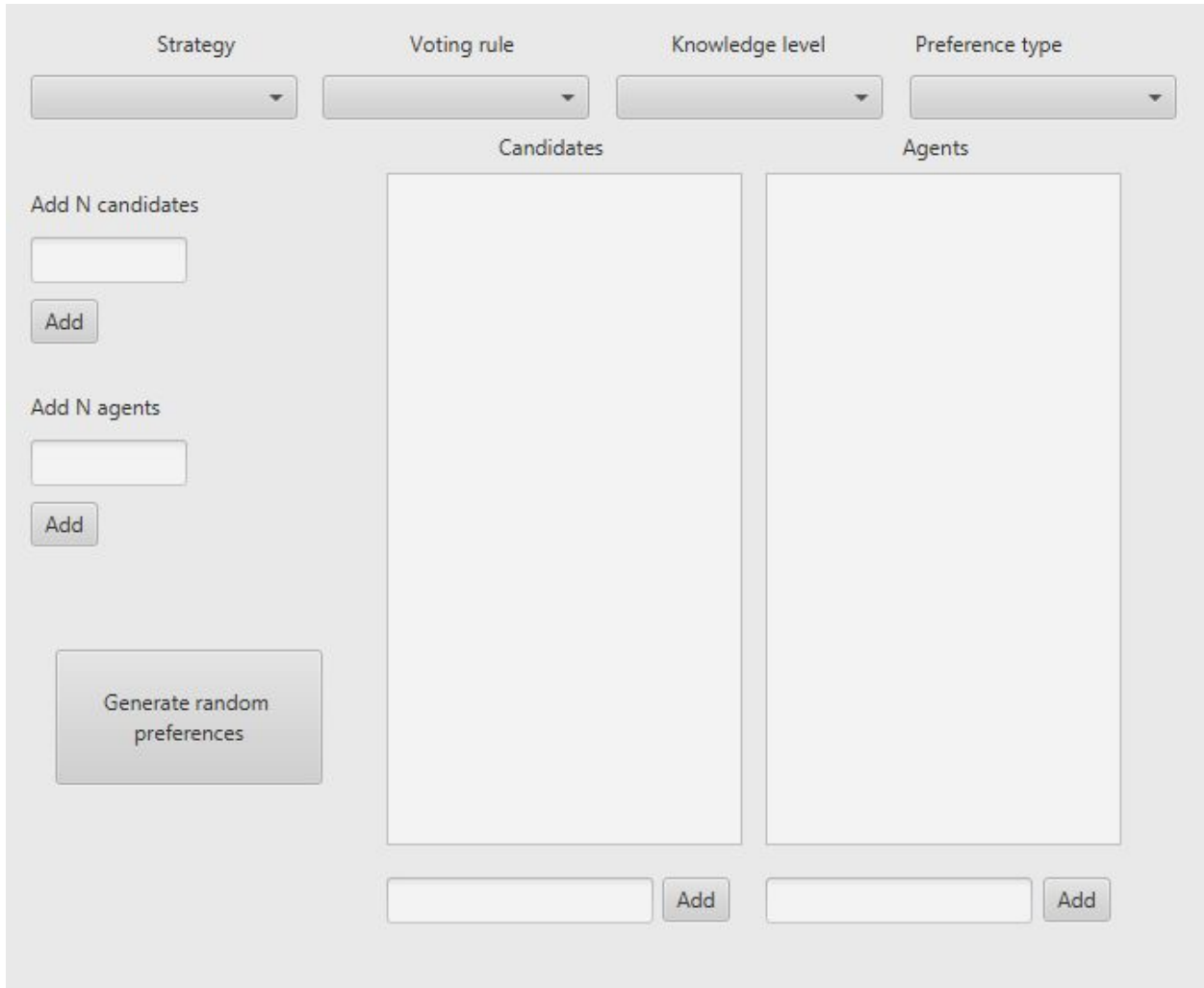
Dans cette section nous démontrons la représentation par défaut de l'évolution de la simulation, représenté dans l'item (3) décrit dans la section précédente. Partons de principe qu'à une itération donnée nous ayons les résultats suivants pour l'élection :

A	54
B	23
C	76
D	23
E	4
F	19

Le graphe donné serait donc :



3.2) Fenêtre de configuration du profil



The interface is a light gray window for configuring a profile. At the top, there are four dropdown menus labeled "Strategy", "Voting rule", "Knowledge level", and "Preference type". Below these, the window is divided into two main sections: "Candidates" on the left and "Agents" on the right. On the far left, there are two input fields for "Add N candidates" and "Add N agents", each followed by an "Add" button. Below these is a large button labeled "Generate random preferences". At the bottom of each column, there is an input field followed by an "Add" button.

fig 3 : fenêtre de configuration du profil

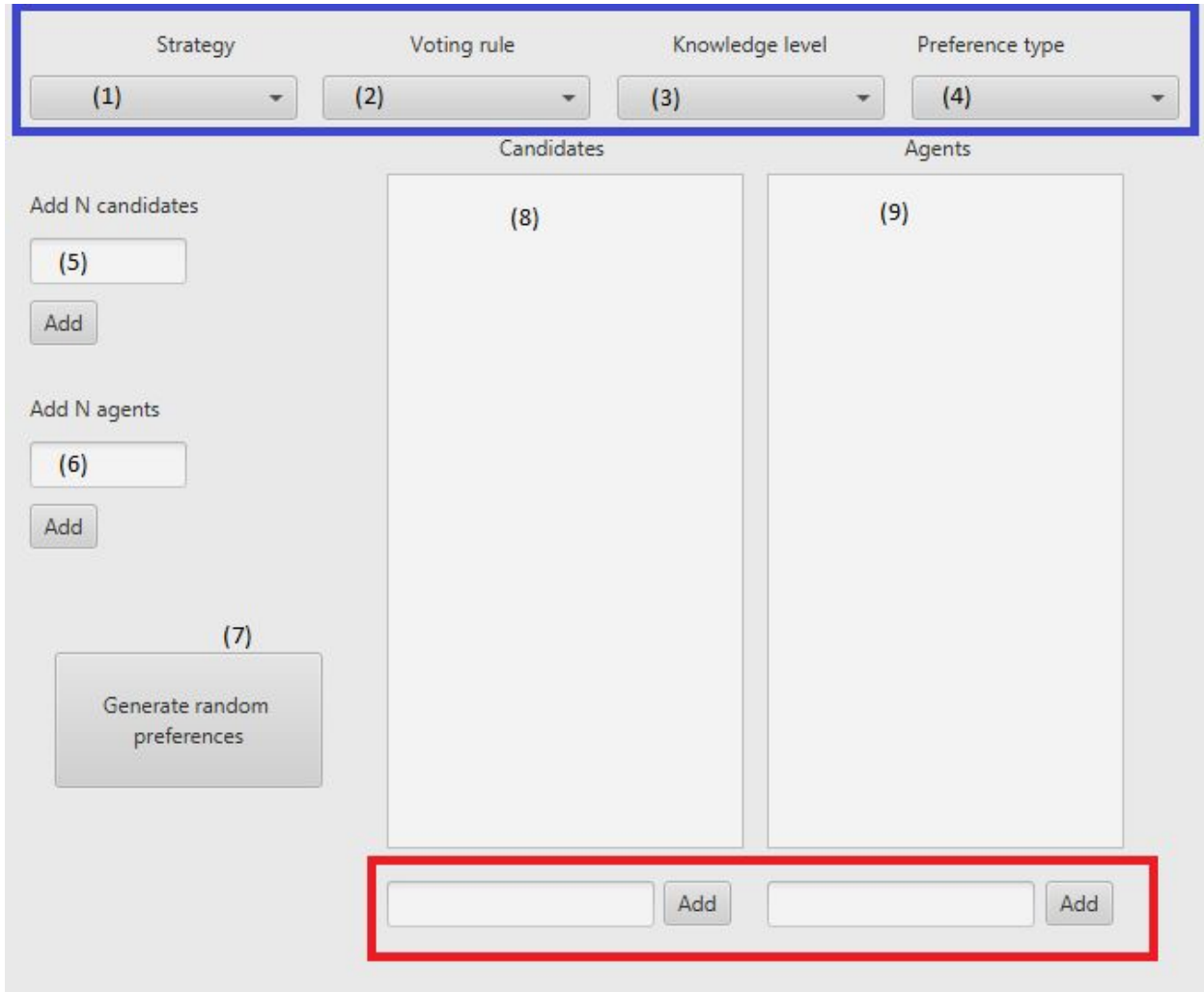


fig 4 : fenêtre de configuration du profil marquée

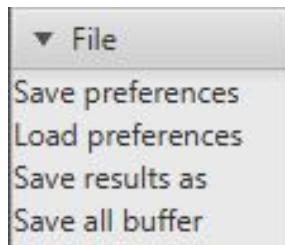
Cette fenêtre est ouverte via la fenêtre principale de l'application (voir section 3.1) en cliquant sur le bouton "Edit Profile". Elle permet à l'utilisateur de facilement configurer la simulation sans avoir à éditer le fichier de configuration. Ci-dessous nous décrivons les différentes parties de cette fenêtre. Très peu d'informations seront données pour les options dans la partie encadrée en bleu. Par défaut, seulement quelques options seront disponibles dans les menus déroulants. Pour plus d'informations, se référer aux sections contenues dans la description de la demande, particulièrement les sections 2.2.b et 2.3.a

- (1) : Choix de la stratégie
- (2) : Choix de la règle de vote
- (3) : Choix du niveau de connaissance

- (4) : Choix du type de préférences. Attention, changer cette option peut potentiellement invalider une configuration déjà existante si le nouveau format de préférences est différent de celui utilisé précédemment
- (5) : Ajoute un nombre N de candidats en leur attribuant des noms génériques
- (6) : Ajoute un nombre N d'agents en leur attribuant des noms génériques
- (7) : Génère des préférences aléatoires pour les agents en se basant sur le format de préférences choisi
- (8) : La liste des candidats. Il est possible de cliquer sur chacun des items afin d'éditer leurs noms. Chaque candidat n'apparaîtra que par son nom
- (9) : La liste des agents. Il est possible de cliquer sur chacun des items afin d'éditer leurs noms et leurs préférences. Chaque agent apparaîtra par son nom et une brève description de ses préférences
- (10) : Cette section permet l'ajout candidat par candidat et agent par agent

3.3) Barre des menus

Dans cette section nous nous intéresserons à la barre des menus introduite dans la section 3.1.



TODO : On doit encore définir ce qu'il y a à mettre dans "views" et pour le "about" ça sera sûrement juste une fenêtre popup donnant quelques informations sur l'application (cadre dans lequel elle a été développée, la licence associée, etc ...)

Dans l'ordre de haut en bas :

- Sauvegarde les préférences de la simulation actuelle
- Charge des préférences sauvegardées précédemment, écrasant celles déjà définies dans l'application en même temps
- Sauvegarde les résultats de la trame actuelle dans un fichier au format .csv
- Sauvegarde l'entièreté du tampon de l'application