INFO-F-106 : PROJET D'INFORMATIQUE
LA RUMEUR COURT...

Gwenaël Joret Jean-Sébastien Lerat Liran Lerman Luciano Porretta
Nikita Veshchikov

version du 22 octobre 2014

Présentation générale

Le projet en quelques lignes

Le but de ce projet est de simuler la propagation d'une rumeur dans un réseau social. Notre réseau, très simpliste, se limitera à représenter les liens d'amitiés existants entre les diverses personnes. Une personne lancera une certaine rumeur, rumeur qui se propagera ensuite de proche en proche à travers le réseau selon certaines règles. Nous considérerons initialement la règle suivante : à chaque unité de temps, chaque personne connaissant la rumeur choisit aléatoirement un de ses amis et la lui transmet. A l'aide du simulateur, nous pourrons par-exemple étudier expérimentalement le temps moyen nécessaire pour que tout le monde apprenne la rumeur.

Le projet est organisé autour de quatre grandes parties. La première partie consistera en la mise en place des structures de données nécessaires à la réalisation de ce projet, et à implémenter une version basique du simulateur. La deuxième partie portera sur l'ajout de différentes fonctionnalités. En particulier, nous allons nous intéresser au contenu de la rumeur, et allons modéliser divers phénomènes de "mutation" de la rumeur lors de sa transmission (une anecdote qui se transmet de bouches à oreilles risque en effet de se voir légèrement modifier au cours du temps ...). La troisième partie ajoutera une interface graphique. La quatrième et dernière partie consistera à ajouter des fonctionnalités supplémentaires au simulateur et/ou à réaliser certaines variantes. Une certaine part de liberté sera ici laissée aux étudiants (et la créativité est encouragée !).

Organisation

Pour toute question portant sur ce projet, n'hésitez pas à contacter par e-mail ou à rencontrer — prenez rendez-vous ! — le titulaire du cours ou la personne de contact de la partie concernée.

Titulaire. Gwenaël Joret – gjoret@ulb.ac.be – O8.111

Assistants.

Partie 1 : Liran Lerman – liran.lerman@ulb.ac.be – N8.212

Partie 2 : Nikita Veshchikov – nikita.veshchikov@ulb.ac.be – N8.213

Partie 3 : Jean-Sébastien Lerat – Jean-Sebastien.Lerat@ulb.ac.be – O8.212

Partie 4 : Luciano Porretta – lporrett@ulb.ac.be – N3.202

Objectifs pédagogiques

Ce projet *transdisciplinaire* permettra de solliciter vos compétences selon différents aspects.

— Des connaissances vues aux cours de programmation, langages, algorithmique ou mathématiques seront mises à contribution, avec une vue à plus long terme que ce que l'on retrouve dans les divers petits projets de ces cours. L'ampleur du projet requerra une analyse plus stricte et poussée que celle nécessaire à l'écriture d'un projet d'une page, ainsi qu'une utilisation rigoureuse des différents concepts vus aux cours.
— Des connaissances non vues aux cours seront nécessaires, et les étudiants seront invités à les étudier par eux-mêmes, aiguillés par les *tuyaux* fournis par l'équipe encadrant le cours. Il s'agit entre autres d'une connaissance de base des interfaces graphiques en `Python 3`.
— Des compétences de communication seront également nécessaires : à la fin des parties 2 et 4, les étudiants remettront un rapport expliquant leur analyse, les difficultés rencontrées et les solutions proposées. Une utilisation correcte des outils de traitement de texte (utilisation des styles, homogénéité de la présentation, mise en page, *etc.*) sera attendue de la part des étudiants. Une orthographe correcte sera bien entendu exigée.
— En plus des deux rapports, les étudiants prépareront une présentation orale, avec *transparents* ou *slides* (produits par exemple avec LaTeX, `LibreOffice Impress`, `Microsoft PowerPoint`), ainsi qu'une démonstration du logiciel développé. À nouveau, on attendra des étudiants une capacité à présenter et à vulgariser leur projet (c'est-à-dire rendre compréhensible leur présentation pour des non informaticiens, ou en tout cas pour des étudiants ayant eu le cours de programmation mais n'ayant pas connaissance de ce projet-ci).

En résumé, on demande aux étudiants de montrer qu'ils sont capables d'appliquer des concepts vus aux cours, de découvrir par eux-mêmes des nouvelles matières, et enfin de communiquer de façon scientifique le résultat de leur travail.

Consignes

A relire avant chaque remise d'une partie du projet !

— L'ensemble du projet est à réaliser en `Python 3`.
— Le projet est organisé autour de quatre grandes parties, auxquelles s'ajoutent la rédaction d'un rapport intermédiaire au milieu du projet, et d'un rapport final en fin de projet.
— En plus de ces quatre parties à remettre, chaque étudiant doit préparer une *démonstration* de cinq minutes, ainsi qu'une *présentation orale* de dix minutes ; celles-ci se dérouleront aux alentours de la fin du projet.
— Chacune des quatre parties du projet compte pour 20 points. Le rapport final (remis en même temps que la partie 4), la démonstration, et la présentation comptent également pour 20 points en tout, ce qui fait un total de 100 points. (Remarque : le rapport intermédiaire sera lui par-contre pris en compte dans la cotation de la partie 2).
— Chacune des quatres grandes parties devra être remise sous deux formes :
  — via l'Université Virtuelle (`http://uv.ulb.ac.be`), qui contient une tâche pour chaque partie ;
  — sur papier, au secrétariat étudiants du Département d'Informatique.
— Après chaque partie, un correctif sera proposé. Vous serez libres de continuer sur base de ce correctif mais nous vous conseillons de plutôt continuer avec votre travail en tenant compte des remarques qui auront été faites.
— Les « Consignes de réalisation des projets » (cf. `http://www.ulb.ac.be/di/consignes_projets_INFO1.pdf`) sont d'application pour ce projet individuel. Vous lirez ces consignes en considérant chaque partie de ce projet d'année comme un projet à part entière. Relisez-les régulièrement !
— Si vous avez des questions relatives au projet (incompréhension sur un point de l'énoncé, organisation, *etc.*), n'hésitez pas à contacter le titulaire du cours ou la personne de contact de la partie concernée, **et non votre assistant de TP**.
— **Il n'y aura pas de seconde session pour ce projet !**

Veuillez noter également que le projet vaudra **zéro** sans exception si :
— le projet ne peut être exécuté correctement via les commandes décrites dans l'énoncé ;
— les noms de fonctions sont différents de ceux décrits dans cet énoncé, ou ont des paramètres différents ;

— à l'aide d'outils automatiques spécialisés, nous avons détecté un plagiat manifeste (entre les projets de plusieurs étudiants, ou avec des éléments trouvés sur Internet).

1 Partie 1 : Création des structures de données et fonctions de base

Les quatre tâches principales de cette partie consistent à réaliser les fonctions permettant de (1) créer le réseau, (2) afficher l'état du réseau, (3) mettre à jour le réseau, et (4) gérer le temps. Vous écrirez vos fonctions dans le fichier `rumorFunctions.py` et le corps de votre programme dans le fichier `rumor.py`. Notez cependant que la fonction principale de votre programme, `main()`, se trouvera elle dans le fichier `rumor.py`.

1.1 Création du réseau

Cette section décrit la création de trois structures de données, la matrice `network`, la liste `names`, et la liste `informedPeople`, ainsi qu'une fonction `inputData()`. Les trois structures de données devront être définies dans le fichier `rumor.py`, et la fonction `inputData()` dans le fichier `rumorFunctions.py`.

La représentation du réseau se fait via une matrice nommée `network` où la i-ème ligne est une liste encodant les amis de la i-ème personne du réseau. Plus précisément, la i-ème ligne contient `True` à la j-ème position si les i-ème et j-ème personnes sont amis, et `False` sinon. (Nous supposerons que la relation d'amitié est symétrique).

La liste `names` contiendra le nom des personnes composant le réseau, chaque nom étant représenté sous forme d'une chaîne de caractères.

La liste `informedPeople` permettra de savoir qui connaît la rumeur. Plus précisément, cette liste contiendra à la i-ème position `True` si la i-ème personne est informée de la rumeur, et l'entier `False` sinon. Au début personne ne connaît la rumeur ; par conséquent, cette liste ne contiendra que des `False` lors de son initialisation.

La fonction `inputData()` demandera à l'utilisateur (à l'aide la fonction `input()`) les informations suivantes :

— le nombre de personnes dans le réseau,
— le nom de chaque personne, et
— les amis de chaque personne.

La fonction `inputData()` devra retourner un tuple consistant en la liste des noms de chaque personne du réseau (qui constituera la liste `names`) et la matrice représentant le réseau (qui constituera la matrice `network`).

La Figure 1 montre un exemple d'exécution de la fonction `inputData()`.

```
Veuillez entrer le nombre de personnes : 3
Veuillez entrer le nom de la 1ère personne : Alice
Veuillez entrer le nom de la 2ème personne : Bob
Veuillez entrer le nom de la 3ème personne : Eve
Alice et Bob sont-ils amis (1 pour oui et 0 pour non) ? : 1
Alice et Eve sont-elles amies (1 pour oui et 0 pour non) ? : 0
Bob et Eve sont-ils amis (1 pour oui et 0 pour non) ? : 1
```

FIGURE 1 – Exemple d'exécution de la fonction `inputData()`.

Vous veillerez à ce que la fonction affiche d'abord le texte "Veuillez entrer le nombre de personnes : " pour demander le nombre de personnes du réseau, puis le texte "Veuillez entrer le nom de la $i$ème personne : " (où i est le numéro de la personne) pour demander le nom de chaque personne du réseau et, enfin le texte "A et B sont-ils (elles) ami(e)s (1 pour oui et 0 pour non) ? : " (où A est le nom d'une personne et B le nom d'une autre personne), pour que l'utilisateur puisse indiquer si A et B sont amis. Comme mentionné précédemment, nous faisons l'hypothèse que si A est ami de B, alors B est également ami A. Il conviendra d'exploiter cette symétrie afin d'éviter de poser des questions inutiles à l'utilisateur.

L'ordre des noms des personnes dans la liste `names` suivra l'ordre des noms donnés lors de l'exécution de `inputData()`.

Dans l'exemple présenté à la Figure 1, à la fin de l'exécution la matrice `network` contiendra `[[True True False], [True True True], [False True True]]`, la liste `names` contiendra `["Alice", "Bob", "Eve"]`, et la liste `informedPeople` contiendra `[False, False, False]`.

1.2 Affichage du réseau

La fonction `printState(names, informedPeople)` prendra en entrée la liste de noms `names` et la liste des personnes qui connaissent la rumeur (`informedPeople`). Cette fonction affichera dans le terminal (via la fonction `print()`) la liste des gens connaissant la rumeur. La Figure 2 illustre le résultat de l'appel à la fonction `printState` pour le réseau construit à la Figure 1 dans le cas où seulement Alice et Bob connaîtraient la rumeur. (Pour rappel, la fonction `printState` devra se trouver dans le fichier `rumorFunctions.py`).

```
Alice connaît la rumeur
Bob connaît la rumeur
Eve ne connaît pas la rumeur
```

FIGURE 2 – Exemple d'exécution de la fonction `printState(names, informedPeople)`.

La fonction `printState(names, informedPeople)` affichera pour chaque personne (et en respectant l'ordre de la liste `names`), une ligne de texte contenant le nom de la personne suivi du texte "connaît la rumeur" si elle connaît la rumeur ou "ne connaît pas la rumeur" si elle ne la connaît pas.

1.3 Mise à jour du réseau

La fonction `update(network, informedPeople)`, qui se trouvera dans le fichier `rumorFunctions.py`, prendra en entrée le réseau (la matrice `network`) et la liste des personnes connaissant la rumeur (la liste `informedPeople`). Pour chaque personne connaissant la rumeur, cette fonction choisira aléatoirement un de ses amis, et l'informera de la rumeur. (Notez qu'il se peut que la personne ainsi choisie connaisse déjà la rumeur !) La fonction retournera le nombre de personnes ayant *appris* la rumeur lors de cette mise à jour. Afin d'éviter toute ambiguïté, précisons que cette mise à jour est bien composée de deux phases distinctes : d'abord on choisit aléatoirement un ami de chaque personne connaissant la rumeur et, une fois tous ces choix effectués, les personnes choisies sont informées de la rumeur. (Le protocole considéré ici est donc *memoryless*, une personne pouvant transmettre la rumeur plusieurs fois au même ami lors d'étapes distinctes).

1.4 Gestion du temps

Nous considèrons que le temps évolue de manière discrète, étape par étape. La fonction `update` sera appelée à chaque nouvelle étape. Il conviendra de demander à l'utilisateur (dans la fonction `main()`) le nombre total d'étapes de la simulation au tout début de l'exécution. Vous ferez ensuite appel à la fonction `inputData()` afin de construire le réseau. Par après, votre fonction `main()` demandera à l'utilisateur le nom de la personne qui "lance" la rumeur, en affichant le texte "Veuillez entrer le nom de la personne à l'origine de la rumeur : ". (On prendra naturellement soin d'encoder le fait que cette personne connaît la rumeur). Une fois cette information connue, vous afficherez l'état initial avant propagation de la rumeur via un appel à la fonction `printState`. Enfin, la simulation proprement dite débutera et la fonction `main()` mettra à jour le réseau étape par étape, en appelant à chaque étape la fonction `update` suivi d'un appel à la fonction `printState` pour afficher l'état courant, jusqu'à atteindre le nombre total d'étapes demandés. (Notons que l'initialisation du réseau n'est pas comptée comme une étape, il y aura donc autant de mise à jour que d'étapes).

La Figure 3 montre un exemple d'une exécution complète du programme.

Pour rappel, la fonction `main()` devra se trouver dans le fichier `rumor.py`. Ainsi, votre programme devra pouvoir être executé via la commande suivante : `python3 rumor.py`.

1.5 Fichiers à soumettre

Toutes les fonctions décrites ci-dessus (à l'exception de la fonction `main()`), ainsi que vos éventuelles fonctions supplémentaires, devront se trouver dans le fichier `rumorFunctions.py`. La fonction `main()` devra être implémentée dans le fichier `rumor.py`. Seuls ces deux fichiers devront être soumis.

```
Nombre d'étapes de la simulation : 4
Veuillez entrer le nombre de personnes : 3
Veuillez entrer le nom de la 1ère personne : Alice
Veuillez entrer le nom de la 2ème personne : Bob
Veuillez entrer le nom de la 3ème personne : Eve
Alice et Bob sont-ils amis (1 pour oui et 0 pour non) ? : 1
Alice et Eve sont-elles amies (1 pour oui et 0 pour non) ? : 0
Bob et Eve sont-ils amis (1 pour oui et 0 pour non) ? : 1
Veuillez entrer le nom de la personne à l'origine de la rumeur :  Alice
Etat initial :
Alice connaît la rumeur
Bob ne connaît pas la rumeur
Eve ne connaît pas la rumeur
Etape 1 :
Alice connaît la rumeur
Bob connaît la rumeur
Eve ne connaît pas la rumeur
Etape 2 :
Alice connaît la rumeur
Bob connaît la rumeur
Eve connaît la rumeur
Etape 3 :
Alice connaît la rumeur
Bob connaît la rumeur
Eve connaît la rumeur
Etape 4 :
Alice connaît la rumeur
Bob connaît la rumeur
Eve connaît la rumeur
```

FIGURE 3 – Exemple d'exécution de la fonction `main()`.

1.6 Conseil d'organisation

Nous vous conseillons de commencer par la création du fichier `rumorFunctions.py` et d'y inclure dans un premier temps uniquement les prototypes des fonctions demandées. Vous pourrez ensuite remplir le corps des fonctions une par une, tout en vous assurant que votre programme (presque vide au début) s'exécute correctement à tout moment.

1.7 Rendu

Personne de contact : Liran Lerman – llerman@ulb.ac.be – N8.212
Remise : Lundi 17 novembre 2014 à 13 heures.
A remettre : Les scripts `rumor.py` et `rumorFunctions.py` en `Python3` :
— Au secrétariat étudiants : une version imprimée de ces scripts ;
— Sur l'UV (`http://uv.ulb.ac.be/`) : une version électronique de ces mêmes scripts.

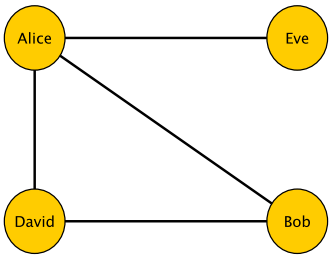

FIGURE 4 – Un réseau.

2 Partie 2 : Fonctionnalités supplémentaires

Cette partie consiste essentiellement à ajouter les fonctionnalités suivantes à votre programme :
— chargement du réseau à partir d'un fichier,
— modification éventuelle de la rumeur lors de sa transmission,
— propagation de la rumeur selon de nouveaux protocoles,
— récupération de paramètres en ligne de commande.

Chaque fonctionnalité sera décrite en détail ci-dessous. Veuillez noter que vous n'allez pas seulement devoir ajouter du code mais également *modifier* le code de la partie précédente.

2.1 Chargement du réseau

Vous avez probablement remarqué lors de la réalisation de la première partie qu'entrer les liens d'amitiés entre les différentes personnes peut prendre beaucoup de temps, surtout si le réseau est grand. Afin d'offrir à l'utilisateur une façon plus pratique d'encoder le réseau, vous ajouterez une fonction `loadNetwork`. Cette fonction prendra en paramètre le nom d'un fichier contenant une description du réseau, et renverra la liste des noms composant le réseau et la matrice représentant le réseau (cf. structures `names` et `network` de la Partie 1).

Le fichier qui décrit le réseau doit être structuré de la façon suivante : chaque ligne commence par le nom d'une personne suivi du symbole `":"`, et ensuite sur la même ligne les noms de tous ses amis (séparés par des virgules). Par exemple, le fichier suivant représente le réseau de la Figure 4 :

```
Alice : Bob, David, Eve
Bob : Alice, David
David : Bob, Alice
Eve : Alice
```

Afin de faciliter la tâche de l'utilisateur, nous ferons l'hypothèse que les espaces sur une ligne du fichier n'ont pas d'importance (il peut y avoir ou non des espaces avant et après les symboles `":"` et `","`). De même, sur une ligne la liste des amis ne doit pas suivre d'ordre particulier. Vous pouvez supposer que le fichier passé en paramètre existe bel et bien, et qu'il a été écrit selon les règles ci-dessus (ceci ne doit donc pas être testé).

2.2 Modification de la rumeur

Quand une personne rapporte une anecdote, il arrive parfois qu'une partie des faits soient oubliés ou légèrement modifiés. Une rumeur qui se propage peut donc se voir modifiée au fil du temps.

La seconde fonctionnalité à ajouter consiste à implémenter ce phénomène. Tout d'abord, nous allons supposer à partir de maintenant que la rumeur est un entier sur 8 bits (1 byte). Lorsqu'une personne

a été mise au courant de la rumeur, il faudra enregistrer la version qu'elle connaît. En particulier, lors d'un affichage de l'état du réseau, il faudra afficher cette information pour les personnes connaissant la rumeur, en binaire ainsi qu'en décimal, comme dans l'exemple suivant :

```
Nom             BIN     DEC
Alice :    00001010      10
Bob :
Charles :  00001011      11
```

Dans l'exemple ci-dessus Alice et Charles connaissent des versions différentes de la rumeur, Bob par-contre ne la connaît pas encore.

La rumeur est modifiée au moment de sa *transmission* d'une personne à une autre, c-à-d quand une personne raconte la rumeur à un de ses amis. La rumeur ne sera pas systématiquement modifiée, cela n'arrivera qu'avec une certaine probabilité p que l'utilisateur pourra spécifier (voir Section 2.3). Concernant la modification de la rumeur, vous implémenterez les trois variantes suivantes (insistons sur le fait qu'une telle modification ne surviendra qu'avec une probabilité p) :

— `incremental` : On choisit aléatoirement d'incrémenter ou de décrémenter la rumeur de 1, chaque option étant equiprobable. Par exemple, si la rumeur était le nombre 5, alors elle deviendra 4 ou 6. Notez qu'il conviendra de respecter l'encodage sur 8 bits de la rumeur ; les opérations d'incrémentation et de décrémentation doivent donc se faire module 2^8.
— `bit flip` : Un des 8 bits de la représentation binaire de la rumeur est choisi aléatoirement (uniformément), et est inversé. Par exemple, si la rumeur était le nombre 5, elle pourrait devenir 4, 7, 69, 13, ... (liste non exhaustive).
— `none` : La rumeur n'est jamais modifiée lors de sa transmission. (Notez que ceci est équivalent à choisir $p = 0$).

Nous allons également considérer trois types de réactions lorsqu'une personne est informée d'une rumeur par un de ses amis, que vous devrez implémenter :

— `stable` : quand une personne apprend la rumeur pour la première fois, elle ne change jamais la version de la rumeur qu'elle connaît, peu importe ce qu'on lui raconte par la suite
— `rewrite` : chaque fois qu'une personne apprend une nouvelle version de la rumeur, elle fait sienne cette nouvelle version
— `mixture` : si Bob apprend d'Alice une certaine version x de la rumeur, alors que Bob connaissait déjà une autre version y de la rumeur, alors Bob remplace sa version par un mélange de x et y comme suit : Tout d'abord, Bob garde tous les bits sur lesquels x et y s'accordent. Ensuite, pour chaque bit où x et y diffèrent, Bob choisit le bit de y avec probabilité 0.9, et celui de x avec probabilité 0.1 (les choix aléatoires étant faits indépendamment les uns des autres). Par exemple, si y vaut 5 (`00000101` en binaire) et x vaut 6 (`00000110` en binaire), alors pour Bob la nouvelle valeur de la rumeur en binaire sera de la forme `000001VW` où `V` et `W` valent soit `0` soit `1`.

Rappelons que la valeur x de la rumeur que Bob *apprend* d'Alice ne correspond pas nécessairement à celle qu'Alice connaît, puisque celle-ci a éventuellement été modifiée lors de sa transmission (voir ci-dessus). En guise d'exemple, supposons qu'on utilise le protocole de modification `incremental` et le protocole de mise à jour `rewrite`. Supposons qu'Alice connaîsse la rumeur, qui pour elle vaut 5, et que Bob lui connaisse 13. Alice raconte 5 à Bob, cependant la modification `incremental` a lieu lors de la transmission, et Bob apprend 4 d'Alice. Ensuite Bob oublie simplement sa version précédente (13) et retient 4 à partir de maintenant pour la rumeur. (Alice, elle, n'a pas changé sa version et garde la valeur 5).

2.3 Arguments en ligne de commande

Afin de faciliter les tests et l'exécution de votre programme, vous ajouterez la possibilité de récupérer des paramètres en ligne de commande sur le terminal (cf. la liste `argv` accessible à l'aide du module `sys`). Pour ce faire, vous utiliserez la librairie `argparse` de Python ; la documentation (avec exemples à l'appui) est disponible à l'adresse suivante : `https://docs.python.org/3/howto/argparse.html`.

Votre programme devra pouvoir se lancer à l'aide de la commande suivante :

```
python3 rumor.py <network filename>
```

Option	Nom	Comportement par défaut	Description
-h	help	–	affiche l'aide pour l'utilisateur (pas de paramètre)
-s	initial person	une personne choisie aléatoirement	paramètre : nom de la personne initiant la rumeur
-r	initial rumor	valeur aléatoire	paramètre : la rumeur de départ (nombre entier entre 0 et $2^8 - 1$)
-t	simulation time	s'arrêter dès que tout le monde connaît la rumeur	paramètre : nombre d'étapes de la simulation
-d	don't tell again	–	si cette option est choisie, chaque personne connaissant la rumeur la transmet à un ami choisi aléatoirement parmi ceux *ne connaissant pas* déjà la rumeur (pas de paramètre)
-m	rumor modification type	`none`	paramètre : type de modification éventuelle de la rumeur lorsqu'elle est racontée (choix : `incremental`, `bitflip`, `none`)
-p	modification probability	0.1	paramètre : probabilité qu'une rumeur soit modifiée lorsqu'elle est racontée (réel entre 0 et 1)
-u	rumor update rule	`stable`	paramètre : règle de mise à jour lorsqu'une personne apprend une nouvelle version de la rumeur (choix : `stable`, `rewrite`, `mixture`)

TABLE 1 – Les options à implémenter.

Notez que l'argument <`network filename`> est obligatoire. D'autre part, nous vous demandons d'implémenter une série d'options, décrites dans la Table 1.

Concernant l'option `-d`, remarquons qu'une personne connaissant la rumeur arrêtera simplement de la transmettre lorsque tous ses amis auront été informés (indifféremment de la source et de la version de la rumeur qu'ils connaissent).

Voici quelques exemples d'appels de votre programme (ceci n'est pas une liste exhaustive) :

```
python3 rumor.py friends.txt -r 15
python3 rumor.py -t 15 friends.txt
python3 rumor.py friends.txt -s Alice -m bitflip -p 0.5 -t 10 -u rewrite -a
python3 rumor.py friends.txt -s Alice -m incremental -a -u mixture -r 40
```

Voici un exemple d'exécution :

```
python3 rumor.py friends.txt -m bitflip -p 0.5
Network Loaded:
Alice: Bob, David, Eve
Bob: Alice, David
Charles: Eve
David: Alice, Bob
Eve: Alice, Charles

Rumor starter: Alice
Initial rumor: 175
```

```
NAME            BIN       DEC
Alice           10101111  175
Bob             -- Does not know --
Charles         -- Does not know --
David           -- Does not know --
Eve             -- Does not know --

Simulation round: 1
1 person just learned the rumor.
NAME            BIN       DEC
Alice           10101111  175
Bob             -- Does not know --
Charles         -- Does not know --
David           -- Does not know --
Eve             10101111  175

Simulation round: 2
2 persons just learned the rumor.
NAME            BIN       DEC
Alice           10101111  175
Bob             -- Does not know --
Charles         10101111  175
David           10100111  167
Eve             10101111  175

Simulation round: 3
1 person just learned the rumor.
NAME            BIN       DEC
Alice           10101111  175
Bob             10100111  167
Charles         10101111  175
David           10100111  167
Eve             10101111  175
 ---- End of simulation ----
```

(Remarque : bien que l'anglais ait été choisi pour l'exemple ci-dessus, son usage n'est aucunement obligatoire).

2.4 Fichiers à soumettre

Les fichiers `rumorFunctions.py` et `rumor.py`. Pour rappel, toutes les fonctions que vous aurez définies doivent se trouver dans le fichier `rumorFunctions.py` (à l'exception de la fonction principale `main()`).

2.5 Rapport

En plus du code, nous vous demandons de rédiger un rapport d'environ 5 pages (page de garde et table des matières non comprises). Ce rapport devra contenir les éléments suivants :

- une page de garde qui reprend vos coordonnées, la date, le titre, *etc.* ;
- une table des matières ;
- une introduction et une conclusion ;
- des sections décrivant ce qui a été réalisés lors des première et deuxième parties ;
- des exemples d'exécution du simulateur ;
- éventuellement des références bibliographiques si applicable.

Le rapport doit détailler les éventuelles difficultés rencontrées, l'analyse et les solutions proposées, ainsi qu'une bonne explication des algorithmes (pseudo-code avec des exemples, pas de code source). Il doit être clair, et compréhensible pour un étudiant imaginaire qui aurait suivi le cours INFO-F-101 (Programmation) mais n'aurait pas fait le projet ; ni trop d'informations ni trop peu. Toutes les parties doivent être détaillées et présentées dans un ordre logique. Expliquez toutes les notions et terminologie que vous introduisez.

Le rapport peut être écrit soit en LaTeX, soit à l'aide des logiciels de traitement de texte `LibreOffice Writer`, `OpenOffice Writer` ou `Microsoft Word`. Il est obligatoire d'utiliser les outils de gestion automatique des styles, de la table de matière et de numérotation des sections. Nous vous conseillons d'utiliser le système LaTeX, très puissant pour une mise en page de qualité. Il est évident qu'une bonne orthographe sera exigée.

Vous trouverez sur l'Université Virtuelle un document donnant des conseils concernant la rédaction de rapports scientifiques.

2.5.1 Vos propositions de fonctionnalités pour la quatrième partie

Dans votre rapport intermédiaire, nous vous invitons à proposer d'autres fonctionnalités / variantes / modifications qui pourraient être implémentées ultérieurement, dans la quatrième partie de ce projet, *en les détaillant dans votre rapport*. Notez que ceci n'est pas obligatoire, et il ne vous en sera pas tenu rigueur lors de la cotation si vous laissez cette section vide. (Cependant, des suggestions particulièrement intéressantes et/ou originales pourraient influencer positivement vos correcteurs—voyez ceci comme une opportunité de points bonus). Lors de la rédaction de l'énoncé de la partie 4, nous tenterons d'incorporer certaines de ces suggestions (sélectionnées sur base de leur pertinence pédagogique : cohérence avec le sujet et difficulté adéquate).

2.6 Rendu

Personne de contact : Nikita Veschikov – nikita.veshchikov@ulb.ac.be – N8.213
Remise : Lundi 15 décembre 2014 à 13 heures.
À remettre : Les scripts `rumor.py` et `rumorFunctions.py` en `Python3`, ainsi que le rapport intermédiaire :

— Au secrétariat étudiants : une version imprimée de ces scripts et du premier rapport intermédiaire ;
— sur l'UV (`http://uv.ulb.ac.be/`) : une version électronique de ces mêmes scripts et du rapport, dans son format source (.doc(x), .odt, ou .tex) et sa version PDF.

3 Partie 3 : Interface graphique

La troisième partie du projet consiste en la mise en œuvre d'une *interface graphique* (*Graphical User Interface* en anglais, d'où GUI) pour votre logiciel qui permettra à l'utilisateur d'interagir avec votre programme de manière plus conviviale. Dans ce but, Python fournit par défaut une bibliothèque nommée `TkInter`. D'autres bibliothèques graphiques pour `Python` existent également, telles que `PyQt`, `PyGTK` ou encore `wxPython`, mais nous vous demandons d'utiliser `TkInter` dans le cadre de ce projet afin de faciliter le travail des correcteurs. Par ailleurs, nous vous demandons qu'au terme de cette troisième partie, votre code gère les erreurs produites lors de l'exécution à l'aide de gestion d'exceptions comme vu au cours de programmation.

3.1 Structure et fonctionnement d'une GUI

Une GUI est typiquement composée d'un ensemble d'éléments nommés *widgets* agencés ensembles. Une fenêtre, un bouton, une liste déroulante, une boîte de dialogue ou encore une barre de menus sont des exemples de widgets typiques. Il peut aussi s'agir d'un *conteneur* qui a pour rôle de contenir et d'agencer d'autres widgets.

Une GUI va typiquement être structurée, de manière interne, sous la forme d'un *arbre* de widgets. Dans `TkInter`, tout programme doit avoir un widget « racine », qui est une instance de la classe `Tkinter.Tk`, créant en fait une fenêtre principale, dans laquelle on peut ensuite ajouter d'autres widgets. À l'appel d'un constructeur de widget, il vous faudra toujours préciser en paramètre quel sera son widget parent (exception faite du widget racine, bien sûr).

Réaliser la structure visuelle d'une GUI (aussi appelée *maquette*) ne constitue que la moitié du travail, il faut que votre code des deux parties précédentes puisse être également lié à cette interface. À l'exécution, une GUI va réagir aux *événements*. Par exemple, un clic sur un bouton. Votre programme ne va donc plus être réduit à se lancer, effectuer des opérations et se terminer. Le principe d'une GUI est d'exécuter une boucle qui va traiter les événement jusqu'á la terminaison de l'application. `TkInter` offre notamment la possibilité de lier le clic sur un bouton à l'appel d'une fonction donnée (voir le paramètre `command` du constructeur de la classe `Tkinter.Button` par exemple). Il vous faudra savamment utiliser ces possibilités pour mettre à jour l'affichage au besoin.

3.2 Exemple de maquette et contraintes

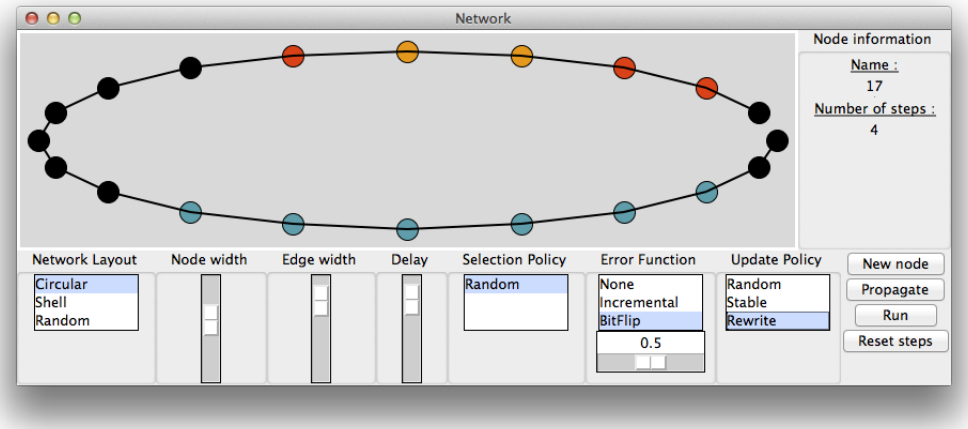

FIGURE 5 – Exemple : canevas de la fenêtre principale de `GUI.py`.

La Figure 5 vous donne un exemple de maquette pour l'interface graphique.

Il doit être possible de fermer votre programme en cliquant sur le bouton de fermeture de la fenêtre. Notez qu'il ne vous est pas imposé de réaliser l'interface en anglais comme sur la Figure 5. Le choix des couleurs, la taille et forme des composantes, leurs positions relatives et les polices de caractères ne sont pas imposées.

Veillez à ce que votre interface graphique soit ergonomique, agréable visuellement et simple d'utilisation.

Remarque sur la performance. Si vous liez une fonction qui nécessite un certain temps de calcul à un bouton de votre interface, celle-ci restera « gelée » après l'événement (clic) jusqu'à ce que la fonction termine son exécution. Dans le cadre de ce projet, nous nous contenterons de ce comportement (et ne le pénaliserons certainement pas lors de la cotation). Pour éviter ce phénomène, il faut exploiter le concept de *multithreading* (qui sort du cadre de ce projet) enseigné aux cours INFO-F201 (Systèmes d'exploitation) et INFO-F202 (Langages de programmation 2).

3.3 Références utiles

La documentation disponible á l'adresse suivante pourra vous aider à développer votre interface graphique avec `TkInter` : `http://wiki.python.org/moin/Tkinter`

3.4 Comportement de GUI

Dans le but d'obtenir une interface plus conviviale, nous allons considérer qu'une rumeur est à présent une couleur représentée sur 24 bits (3 bytes), en représentation *RGB*. Il conviendra d'adapter vos fonctions de la partie 2 qui supposaient un encodage sur 8 bits à un encodage sur 24 bits (par-exemple, incrémentation & décrémentation de la rumeur se font maintenant modulo 2^{24}). Nous demanderons également de respecter les règles suivantes :

Le canvas du réseau Lorsque la souris survole une personne, celle-ci est mise en évidence à l'aide d'une bordure rouge. Il en va de même pour les liens. Ceux-ci ne possèdent pas de bordure mais deviennent rouge lors du survol par la souris. Un double-clic permet de supprimer l'élément survolé. De plus, le clic droit de la souris sur une personne offre la possiblité de modifier la rumeur en affichant une boîte de dialogue *tkinter.colorchooser*. Enfin, l'ajout d'un lien d'amitié entre deux personnes se fait via le mécanisme *drag-and-drop*. C'est-à-dire que l'utilisateur clique sur une première personne et maintient le bouton gauche enfoncé. Il déplace ensuite le curseur sur la seconde personne et relâche enfin le bouton. Cela créera un lien d'amitié entre les deux personnes sélectionnées. Afin de guider l'utilisateur, un lien de couleur bleue entre la personne de départ et le curseur sera dessiné tant que l'utilisateur n'a pas fini de sélectionner la seconde personne.

Le panneau d'information Le panneau d'information affiche le nom de la personne survolée ainsi que le compteur du nombre d'étapes.

Le menu de configuration Le menu de configuration se compose de plusieurs éléments :

1. une liste pour choisir l'algorithme de disposition des personnes
2. un curseur de défilement pour choisir la taille (compris entre $[1;50]$, 20 par défaut) des personnes
3. un curseur de défilement pour choisir l'épaisseur (compris entre $[1;10]$, 2 par défaut) des liens
4. un curseur de défilement pour choisir le délai (compris entre $[0;3]$, $\frac{1}{2}$ seconde par défaut) entre deux étapes
5. une liste pour sélectionner l'algorithme de choix du voisin à qui transmettre la rumeur
6. une liste pour sélectionner le type de modification de la rumeur lors de sa transmission
7. une liste pour sélectionner le protocole de mise à jour de la rumeur lorsqu'une personne reçoit une nouvelle version de celle-ci
8. un curseur de défilement pour choisir la probabilité d'erreur (compris entre $[0;1.0]$, $\frac{1}{2}$ par défaut)

9. un bouton pour ajouter une nouvelle personne ouvrant une boîte de dialogue qui demande le nom de cette personne. Lors de cette opération, une nouvelle personne est ajoutée au réseau et sa bordure est tracée en rouge à l'aide de pointillés
10. un bouton qui lance l'exécution d'une étape supplémentaire de la simulation
11. un bouton qui va ouvrir une boîte de dialogue demandant à l'utilisateur d'entrer un nombre k et lance ensuite l'exécution de k étapes supplémentaires

3.5 Dessin sur un canevas `TkInter`

Nous vous fournissons ici un algorithme qui vous permettra de dessiner des rectangles et des cercles dans un canevas `TkInter` (c'est-à-dire une instance de la classe `Tkinter.Canvas`).

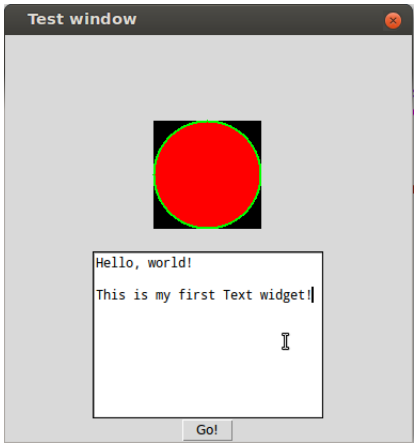

FIGURE 6 – Exemple de maquette *Tkinter*.

```
myFrame = Frame(masterWidget) # création de la Frame
myCanvas = Canvas(myFrame, height=500, width=500) # définit la taille du canevas

# Dessine un rectangle noir de taille 100x100
# Le coin supérieur gauche se trouve en coordonnées (50, 80)
myCanvas.create_rectangle(50, 80, 150, 180, fill="black")

# Dessine un cercle (rouge avec un contour vert) inscrit dans le carré
myCanvas.create_oval(50,80,150,180, outline="green", fill="red", width=2)

# Place le frame et le canevas sur le root widget
# pour placer les objets sur un frame on peut aussi utiliser ma méthode grid
myCavnas.pack()
myFrame.pack()
```

Remarquez que les rectangles sont définis à l'aide de deux points : le coin supérieur gauche et le coin inférieur droit. Les ovales (ainsi que les cercles et les ellipses) sont définis à l'aide des rectangles (l'ovale est alors inscrit dans un rectangle).

De la même manière, on peut ajouter des objets de types `Button`, `Label`, `Menu`, `Text` (et beaucoup d'autres) sur un `Frame`. La Figure 6 montre un exemple d'une fenêtre créée à l'aide de `TkInter`. Mous vous invitons à explorer différents éléments graphiques qui existent dans `TkInter` afin de choisir les éléments qui conviennent le mieux pour votre interface. Il est également préférable d'utiliser des couleurs dans votre interface graphique.

3.6 Gestion d'exceptions

Pour cette phase, la gestion d'exceptions doit être implémentée. Nous nous attendons donc à ce que vous utilisiez pleinement et intelligemment les instructions `try-except-finally` ainsi que la commande `raise` pour gérer proprement les erreurs pouvant survenir lors de l'exécution de votre programme. En

particulier, nous nous attendons à ce que les erreurs relatives aux données erronées donnent lieu à l'affichage de boîtes de dialogue qui décrivent ce qu'il s'est passé (utilisez pour cela le module `messagebox` fourni avec `TkInter`). Vous trouverez un exemple sur la Figure 7.

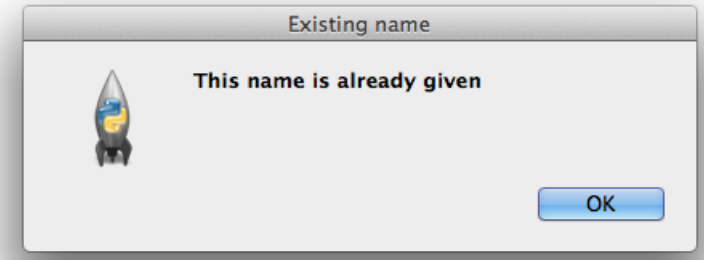

FIGURE 7 – Boîte de dialogue d'erreur avec *messagebox.showerror("Existing name","This name is already given")*.

3.7 Utilisation de classes

Dans la mesure du possible, nous souhaitons que vous fassiez un usage judicieux de classes (et donc exploiter les constructeurs, les attributs et les méthodes) pour structurer votre code. Nous vous demandons de créer au moins les classes suivantes :

- `GUI` - classe qui encapsule l'interface graphique ;
- `NetworkFrame` - classe du canvas qui va contenir le dessin du réseau ;
- `Person` - classe graphique qui représente une personne du réseau ;
- `Link` - classe graphique qui représente un lien entre deux personnes du réseau ;
- `PlatformUtils` - classe fournie qui possède deux méthodes afin d'identifier les boutons gauche et droit de la souris.

Il est également attendu de créer un fichier par classe.

3.8 Exécution du programme

Nous demandons que votre interface graphique soit lancée via l'exécution du fichier `GUI.py` (contenant la classe `GUI`), qui ne prendra aucun paramètre en ligne de commande. Notez cependant que votre programme console devra toujours fonctionner. Pensez à modulariser votre code de manière appropriée. (Pour rappel, aucun code d'exemple issu d'Internet, et/ou d'autres projets et/ou de livres n'est autorisé !)

3.9 Rendu

Personne de contact : Jean-Sébastien Lerat– Jean-Sebastien.Lerat@ulb.ac.be – O8.212
Remise : Lundi 2 mars 2015 à 13 heures.
À remettre : Les scripts et les autres fichiers nécessaires au fonctionnement du projet (par exemple des images ou autres fichiers texte ou scripts python que vous utiliseriez) :

- Au secrétariat étudiants : une version imprimée de vos scripts Python ;
- Sur l'UV (`http://uv.ulb.ac.be/`) : une version électronique de vos fichiers.

4 Partie 4 : Ajout de fonctionnalités supplémentaires

Aperçu

Dans cette quatrième et dernière partie du projet, vous devrez ajouter des fonctionnalités supplémentaires à votre logiciel, que vous choisirez parmi une liste de propositions que nous vous communiquerons ultérieurement. Chaque fonctionnalité correctement implementée rapportera un certain nombre de points ; vous devrez donc en implémenter plusieurs pour espérer atteindre les 20 points que vaut cette dernière partie du projet.

L'énoncé de cette quatrième partie sera disponible au début du second semestre.

4.1 Rapport

En plus du code, on demande un rapport de 9 à 10 pages (page de garde et table des matières non comprises), avec au minimum 4000 mots. Comme pour le premier, ce rapport doit comprendre :

— Une page de garde qui reprend vos coordonnées, la date, le titre, etc. ;
— La table des matières ;
— Une introduction et une conclusion ;
— Des sections, comprenant ce qui a été fait dans le projet, donc les parties 1, 2, 3 et 4 ;
— Des références si nécessaire.

Il peut aussi comprendre des annexes (facultatif) si vous pensez qu'elles sont nécessaires pour des explications supplémentaires (comme des parties de code). Les annexes ne sont pas toujours lues, donc les informations indispensables à la compréhension du projet ne peuvent pas s'y trouver. Les annexes ne comptent pas non plus dans les 9–10 pages de rapport.

Les instructions données pour le premier rapport s'appliquent également ici.

Vous pouvez reprendre des parties du premier rapport, mais faites attention à la continuité de l'ensemble du rapport. Il ne suffit pas de rajouter les parties 3 et 4 au document précédent, il faut retravailler le rapport dans son ensemble. Si dans les parties 1 et 2 il y avait des erreurs de contenu ou autre (orthographe, typographie. . .), elles doivent être corrigées.

Dans l'introduction et la conclusion, tout le projet doit être présenté avec ses objectifs et résultats.

Dans les sections, tout ce que vous avez fait doit être expliqué. Vous devez décrire le projet dans son ensemble, et non partie par partie. Ainsi, évitez les noms de section tels que « Partie 1 », « Partie 2 », *etc.*

On rappelle que le rapport doit être compréhensible par tout étudiant (imaginaire) qui aurait suivi le cours INFO-F101 (Programmation) mais n'aurait pas fait le projet.

Pour des conseils sur la rédaction d'un bon rapport, nous vous renvoyons à nouveau vers le document en annexe.

4.2 Présentation

À coté d'un rapport écrit qui explique un projet en détails, il est souvent demandé, dans les travaux scientifiques, une petite présentation orale qui résume le projet. On vous demande donc une présentation de 10 minutes (pas plus !), où, à l'aide de transparents, vous nous expliquez votre projet.

La présentation ne doit pas aller dans les détails, mais expliquer l'idée générale et les objectifs du projet, les méthodes utilisées et les résultats obtenus. Des exemples et des figures aident beaucoup dans un exposé oral.

Pour des conseils sur la réalisation d'une bonne présentation, voir de nouveau le document en annexe. On remarque juste que pour un exposé de 10 minutes, il faudrait compter maximum 10–12 transparents (la règle empirique est 1 transparent par minute).

Pour vous aider à préparer votre présentation, voici quelques éléments que nous voudrons y trouver :

1. Structure de l'exposé :
   — Une introduction de la présentation (ce dont on va parler) ;
   — Une introduction du projet (les objectifs) ;
   — Différentes sections clairement identifiables, expliquant la contribution, les difficultés rencontrées, les solutions apportées, éventuellement les problèmes non résolus, etc. ;
   — Une conclusion (ce que l'auditoire doit retenir).

2. Qualité du support :
   — Des transparents pas trop chargés, sans code source. Éviter les détails de trop bas niveau (nom/signature de fonctions, de variables. . .) ;
   — Des illustrations, des schémas, des diagrammes (découpe du programme, lien entre les parties, schémas techniques. . .) ;
   — Une orthographe soignée, un vocabulaire adéquat, une uniformité d'aspect entre les pages ;
   — Des transparents numérotés.
3. Qualité de l'exposé :
   — Respect des contraintes temporelles ;
   — Pas trop d'hésitation, pas devoir chercher ses mots, mais pas non plus réciter un texte par cœur ;
   — Présence, volume de la voie, articulation, regarder l'auditoire...

4.3 Rendu

Personne de contact : Luciano Porretta – lporrett@ulb.ac.be – N3.202
Remise : Lundi 30 mars 2014 à 13 heures.
À remettre : L'ensemble des scripts en `Python3` nécessaires au fonctionnement de votre projet, ainsi que le rapport final :

— Au secrétariat étudiants : une version imprimée de ces scripts, et du rapport final ;
— Sur l'UV (`http://uv.ulb.ac.be/`) : une version électronique de ces mêmes scripts et du rapport final, dans son format source (.doc(x), .odt, ou .tex) et sa version PDF.