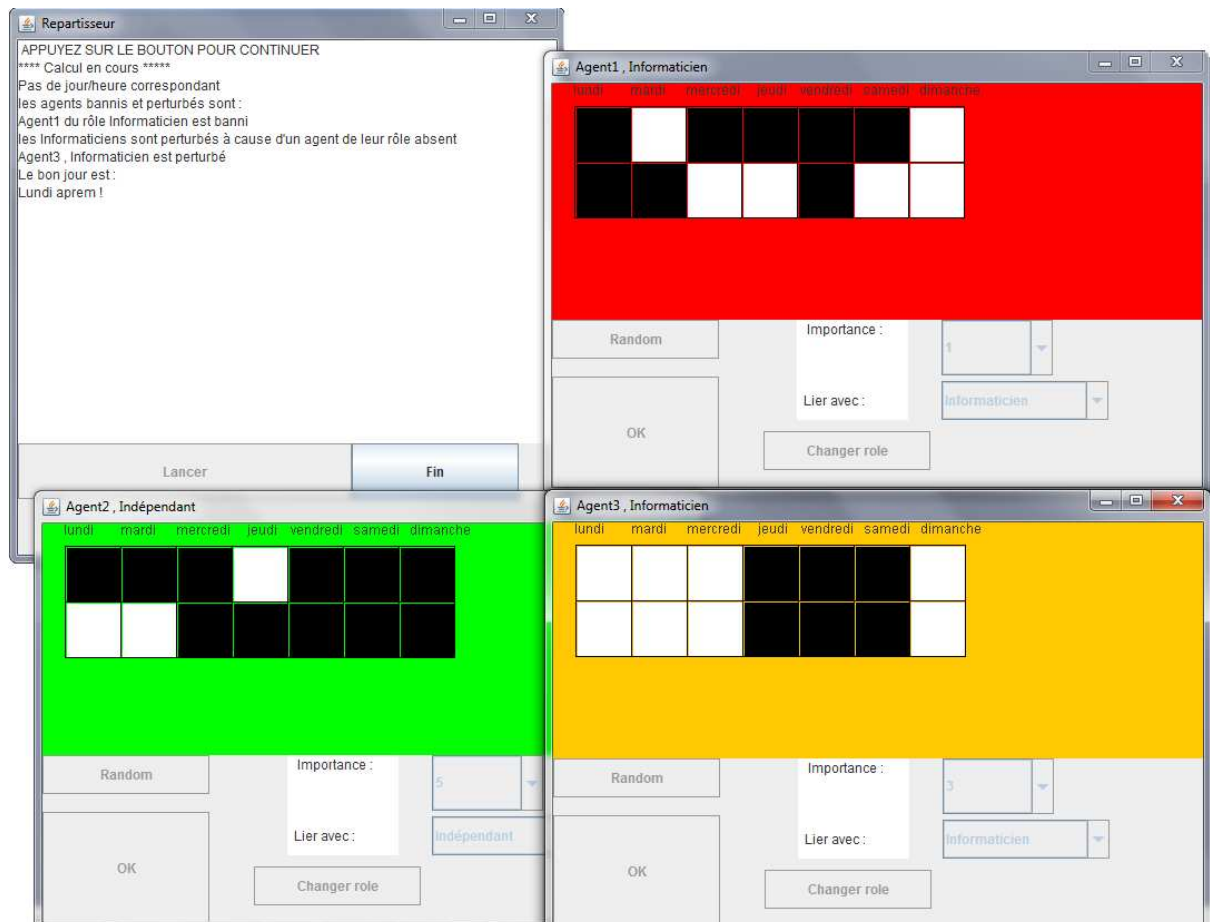




MAURIN Benjamin
GINER Steve
2010/2011
Groupe 45
Q1

Rapport de projet : Apprentissage à la programmation multi-agents



Tuteur de projet : MICHEL Fabien

Remerciements

Tout d'abord, nous tenons à remercier notre tuteur de projet MICHEL Fabien. Son aide a été précieuse voir indispensable à la réalisation de ce projet. Il a toujours été présent pas l'intermédiaire du forum pour répondre à nos question et nous fournir de l'aide. Il n'a pas hésité à modifier MadKit pour nous permettre d'avancer et de finaliser le projet.

Nous voulons également remercier M.BETAILLE pour ses cours de Java qui nous ont bien aidés, notamment pour l'interface du projet.

Enfin, nous remercions nos camarades qui travaillaient sur la même thématique de projet avec qui nous avons pu discuter des différentes fonctionnalités et possibilités de MadKit.

Table des Matières

I. Glossaire	4
II. Introduction	5
II.1. Introduction générale.....	5
II.2. Présentation générale du projet	5
II.3. Méthodes de fonctionnement	6
II.3.1. Fonctionnement avec le tuteur	6
II.3.2. Fonctionnement à l'extérieur du groupe	6
II.3.3. Fonctionnement au sein du groupe.....	6
III. Analyse	8
III.1. Analyse Première version	8
III.2. Analyse seconde version.....	10
III.3. Analyse version finale.....	12
III.4. Schémas UML de la version finale.....	14
III.4.1. Diagramme des classes persistantes.....	14
III.4.2. Diagramme de cas d'utilisation (use case).....	15
IV. Description du code source.....	16
IV.1. Interface	16
IV.1.1. Interface Repartisseur	16
IV.1.2. Interface des agents de contraintes	17
IV.2. Création du tableau de contraintes.....	21
IV.2.1. Initialisation tableau de contraintes personnelles	21
IV.2.2. Création du tableau de contraintes de groupe.....	23
IV.3. Mise en commun des contraintes.....	24
IV.4. Prise de décision du rendez-vous.....	26
IV.4.1. Il existe une solution possible.....	26
IV.4.2. Pas de solution, bannissement d'agent(s) pour arriver à un accord.....	27
V. Conclusion	29
V.1. Bilan	29
V.2. Ce que ce projet nous a apporté	29
V.3. Améliorations possibles du logiciel	30
VI. Bibliographie et logiciels utilisés.....	31
VI.1. Bibliographie	31
VI.2. logiciels utilisés	31
VII. Annexes	32
VII.1. Liens utiles	32
VII.2. Exemple d'utilisation	32

I. Glossaire

Multi-agents : Type de programmation mettant en scène différents agents pour parvenir à résoudre un problème donné. Il met en place un environnement contenant des relations, des opérateurs et des agents.

MadKit : Plate-forme évolutive écrite en Java et construit sur le modèle d'organisation AGR (Agent / Groupe / Rôle): les agents sont situés dans des groupes et jouent des rôles. Cette plate-forme de programmation nous permet de créer un programme en multi-agents.

Agent : Entité virtuelle autonome qui interagit avec son environnement. Ici, un agent appartient à un groupe et un rôle et peut communiquer avec les autres agents de la communauté.

AgentAddress : Classe de MadKit désignant l'adresse d'un agent unique en fonction son groupe et de son rôle. On peut grâce à cette adresse utiliser des méthodes en ciblant un agent précis. Par exemple, la méthode `sendMessage(AgentAddress a, Message m)` enverra le message `m` à l'agent ayant l'AgentAddress `a`.

broadcastMessage(String c, String g, Message m) : fonction de MadKit permettant d'envoyer le message `m` à tous les agents de la communauté `c` et du groupe `g`.

II. Introduction

II.1. Introduction générale

L'intelligence artificielle prend de plus en plus d'ampleur dans le monde d'aujourd'hui. Le multi-agents est une méthode de programmation pour créer des intelligences artificielles.

Dans le cadre du projet de seconde année à l'IUT informatique de Montpellier, nous avons réalisé un exemple de logiciel fonctionnant en multi-agents en utilisant la plate-forme MadKit. Le projet consiste à la création d'un logiciel fonctionnant avec MadKit qui nous permet d'apprendre les bases du multi-agents et d'aider les prochains utilisateurs de MadKit à mieux les comprendre. Nous devons donc choisir un problème à résoudre en multi-agents.

II.2. Présentation générale du projet

Après discussion entre nous puis avec notre tuteur (M.MICHEL), nous avons décidé de créer un logiciel permettant de trouver un rendez-vous entre différents agents en fonction de leurs contraintes. L'utilisateur choisit un nombre d'agent ayant chacun des contraintes puis le programme trouve le rendez-vous le plus intéressant.

Au fur et à mesure de l'avancement du projet, nous avons créé de nouvelles versions ayant de nouvelles fonctionnalités.

Dans la version finale, voici toutes les fonctionnalités de notre logiciel :

- Entrée des contraintes par l'utilisateur par l'intermédiaire d'une interface graphique.
- Possibilité d'entrer les contraintes aléatoirement.
- Bannissement d'agent(s) pour trouver une solution s'il n'en existe pas de base de façon optimale.
- Système d'importance des agents (rentrée par l'utilisateur) permettant de hiérarchiser les agents, cette hiérarchie est prise en compte dans le calcul.
- Système de rôles pour les agents prit en compte dans le calcul de bannissement.

Vous pouvez télécharger la version finale en suivant ce lien :

<http://www.madkit.org/démonstration/RendezVous/>

II.3. Méthodes de fonctionnement

II.3.1. Fonctionnement avec le tuteur

Notre tuteur ne travaillant pas à temps plein à l'IUT et étant donné qu'il avait plusieurs groupes de projet à sa charge, le nombre de possibilités de rendez-vous avec lui était plutôt restreint. Cependant, la communication avec notre tuteur a été importante. En effet, en plus de la communication par E-mail qui nous permettait de fixer les quelques rendez-vous, M.MICHEL a mis à notre disposition des topics sur le forum du site officiel de MadKit dans lequel nous pouvions poser des questions techniques.

Ceci nous a permis de résoudre des problèmes de programmation liés à MadKit. Ce système de communication nous a, entre autres, facilité la création d'un exécutable.

II.3.2. Fonctionnement à l'extérieur du groupe

D'autres groupes travaillaient sur des projets à programmer avec MadKit. Avec eux, nous avons pu discuter à l'IUT de certaines fonctions de MadKit.

Nous avons pu, par exemple, grâce à BRIVAL Stephan en apprendre plus sur la classe AgentAddress.

II.3.3. Fonctionnement au sein du groupe

Le sujet étant libre nous devions nous fixer notre propre cahier des charges. Nous avons décidé de fonctionner avec plusieurs versions.

Pour chaque nouvelle version, nous nous réunissions pour conceptualiser un nouveau problème et une nouvelle solution (nouveau cahier des charges), puis nous nous répartissions les tâches pour effectuer chacun une partie du travail.

Enfin, nous nous réunissions pour mettre en commun, effectuer une batterie de test et corriger les éventuelles erreurs.

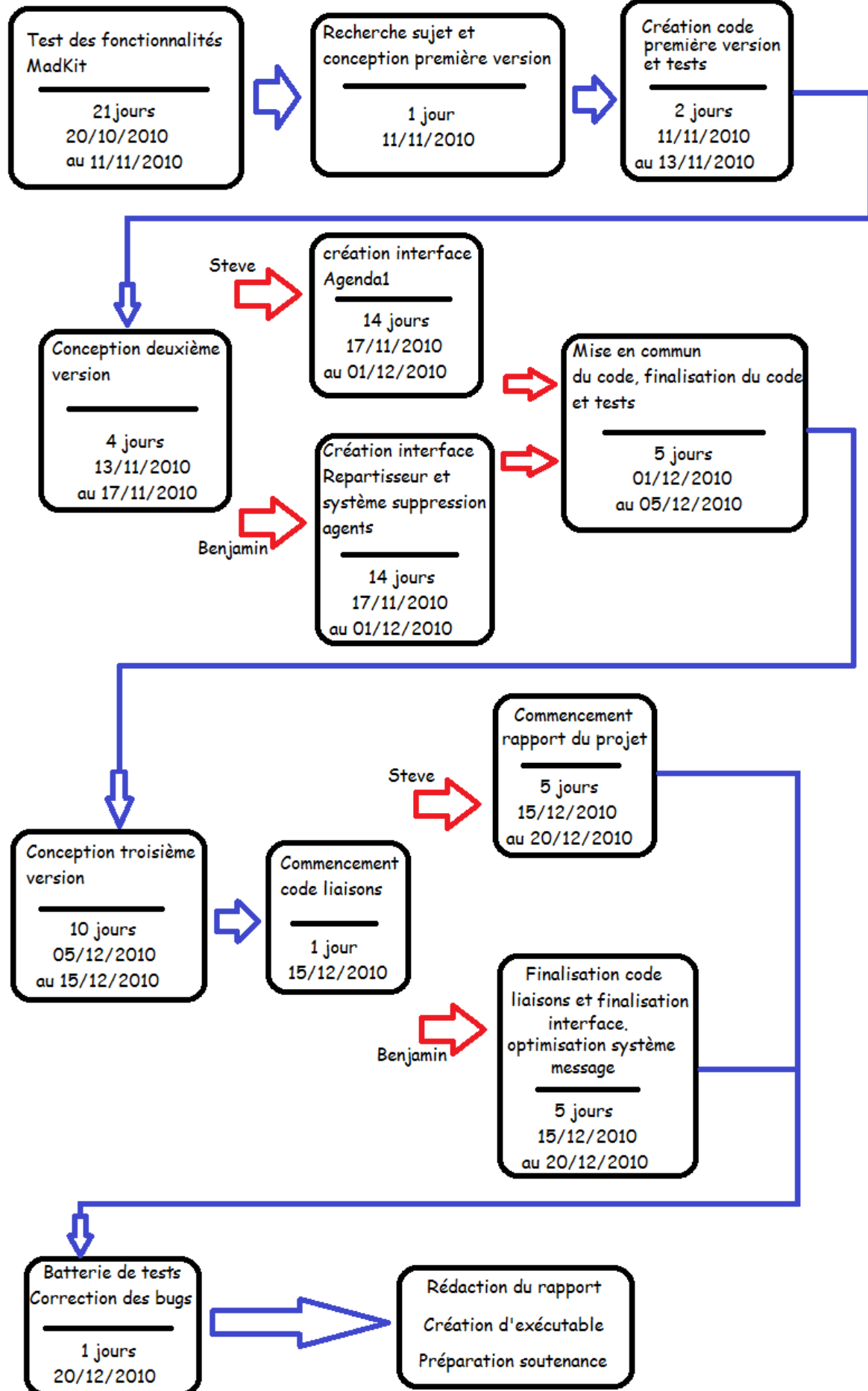
Nous avons ainsi réalisé trois versions principales chacune étant divisible en sous-versions.

Les différentes versions étaient enregistrées sur nos ordinateurs personnels après chaque mise en commun. Elles étaient aussi enregistrées sur nos boîtes E-mail pour éviter toute éventuelle perte de données.

Étant dans la même classe, la communication entre membres du groupe était plutôt aisée. Lorsque l'un de nous avait une idée pour le projet, il pouvait en parler à l'IUT et l'approfondir avec l'autre membre du groupe pendant, par exemple, la pause déjeuner.

Nous utilisions aussi pour communiquer les SMS et E-mails afin de s'informer sur l'avancement de nos travaux respectifs pour permettre de mieux fixer la prochaine mise en commun.

A chaque mise en commun, nous remplissions le blog demandé pour la partie communication et nous remplissions également un carnet de bord décrivant l'avancement du projet pour faciliter la rédaction du rapport.



III. Analyse

Dans cette partie, la conceptualisation de chaque version sera analysée et détaillée. Nous montrerons ainsi l'évolution de notre vision et de notre compréhension d'un logiciel multi-agents.

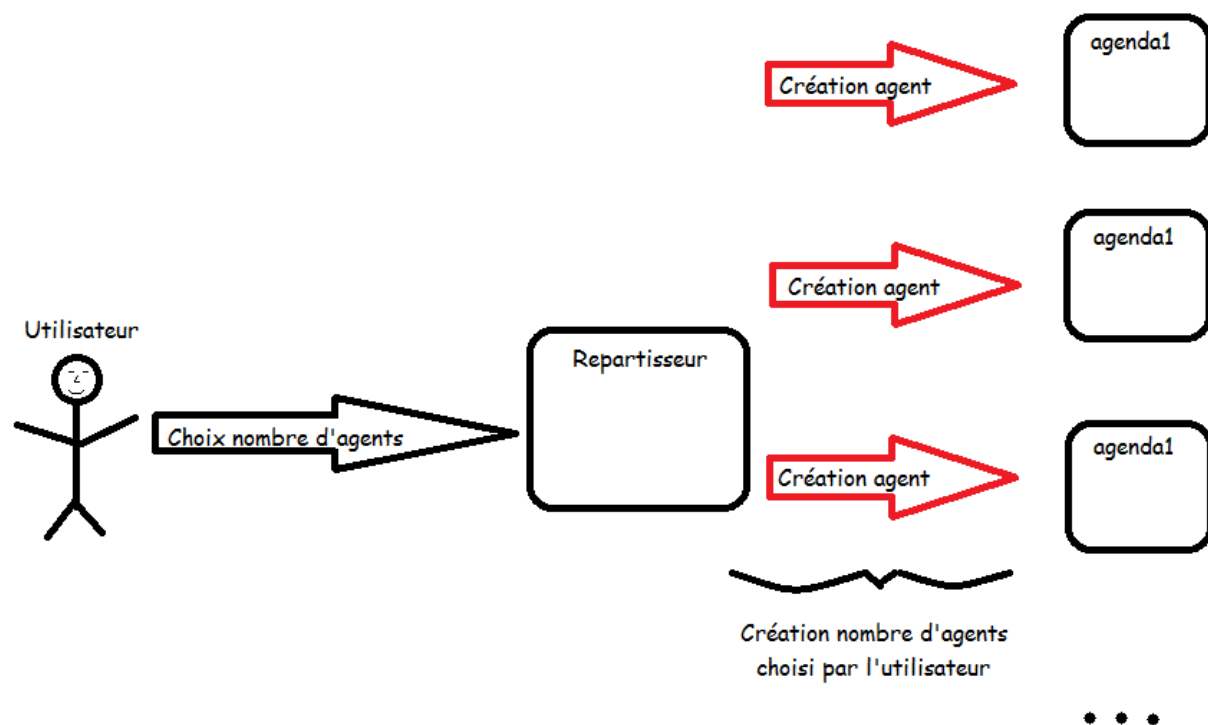
III.1. Analyse Première version

Lors de la création de la première version, on s'était concentré sur la programmation de la recherche du jour adéquat.

Pour ce faire, nous avons l'idée d'avoir :

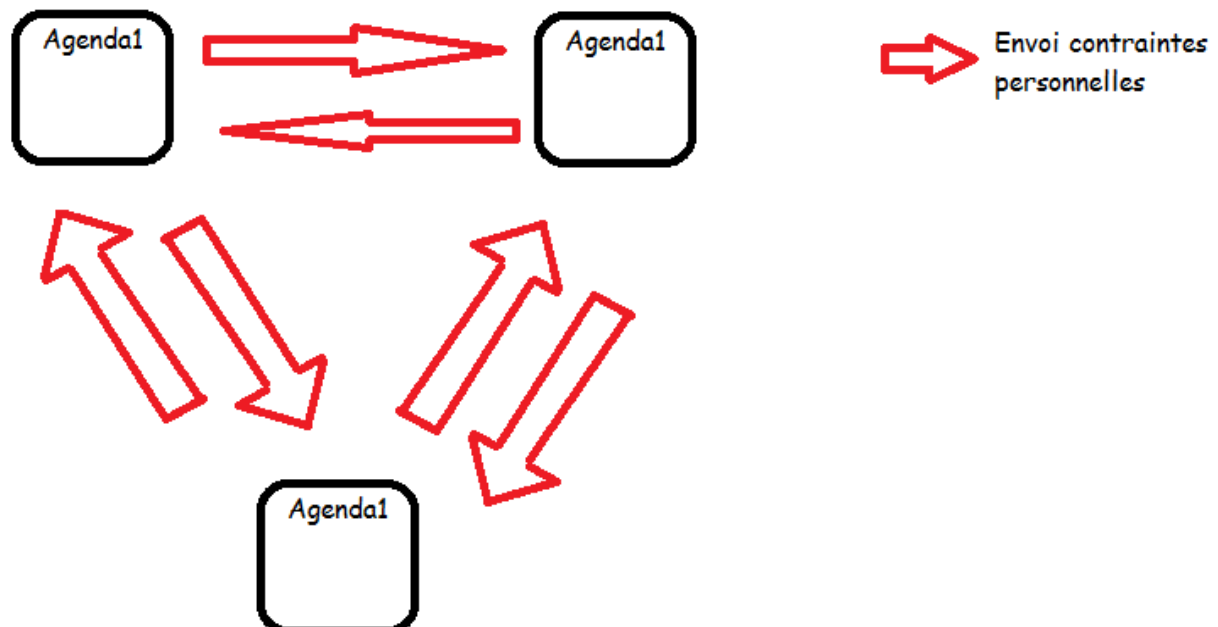
- Un agent qui crée les agents qui ont des contraintes et réceptionnent la somme des contraintes de la communauté d'agents.
- Des agents représentant les agendas de personnes souhaitant un rendez-vous.

L'utilisateur choisissait le nombre d'agent souhaitant un rendez-vous.

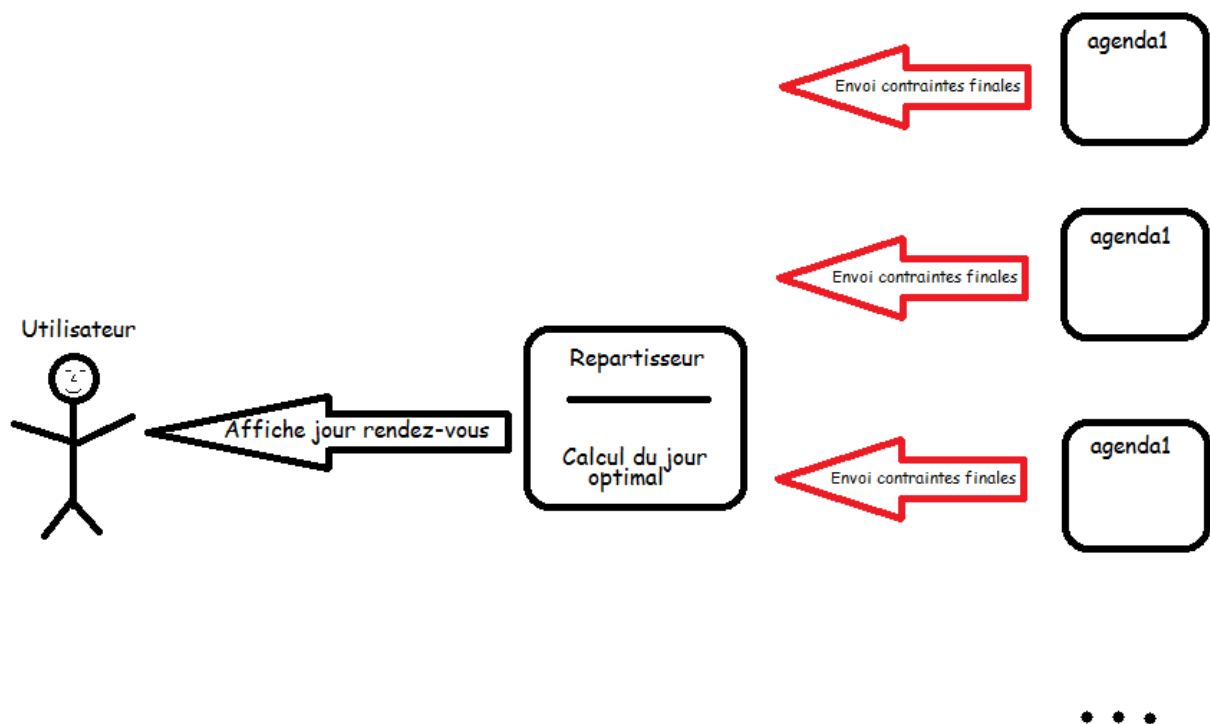


Une fois les agents créés, des contraintes aléatoires leurs sont attribuées. Les contraintes sont réparties sur une durée de une semaine, chaque jour étant divisé en deux périodes : matin et après-midi.

Une fois les contraintes rentrées, chaque agent Agenda1 envoie ses contraintes personnelles à tous les autres agents Agenda1. Chaque agent réceptionne les contraintes des autres agents et calcul la somme des contraintes de la communauté d'agent Agenda1.



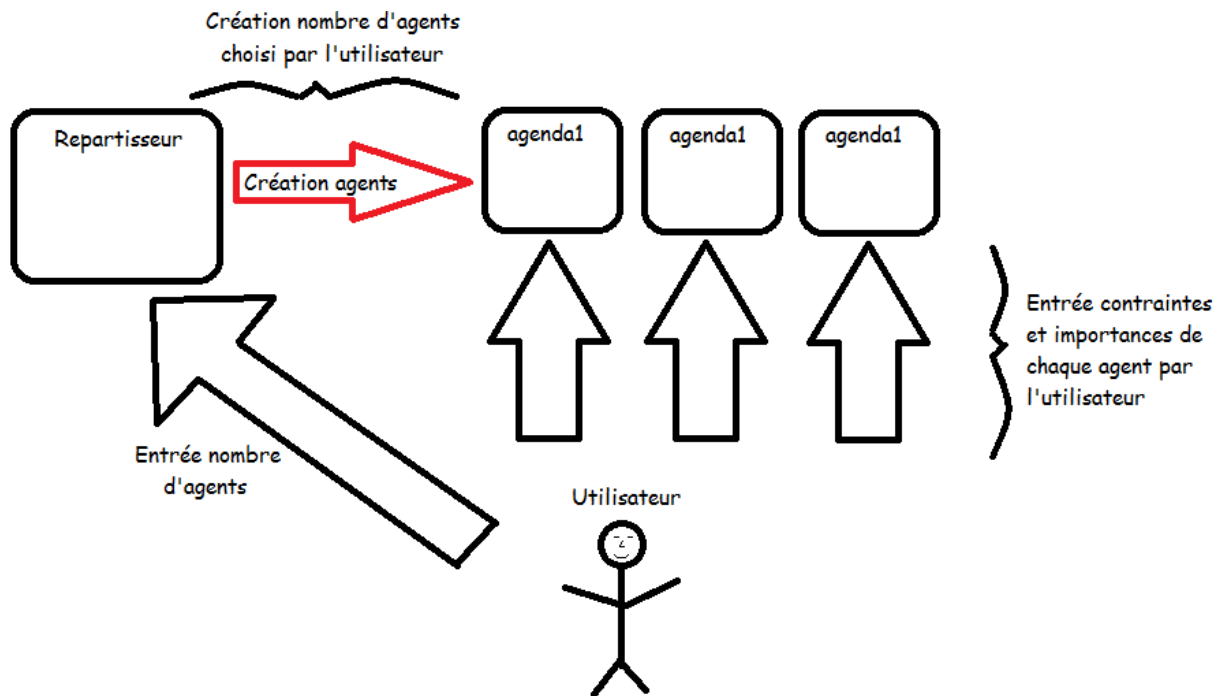
Ensuite, les agents Agenda1 envoient à l'agent Repartisseur les contraintes qu'il analysera pour le jour le plus tôt s'il y en a un possible. Si il n'y en pas le message « pas de jour correspondant » s'affiche à l'écran.



III.2. Analyse seconde version

Pour la seconde version nous nous sommes focalisés sur les choix de l'utilisateur. Il devait choisir :

- Le nombre d'agents
- Les contraintes de chaque agent
- L'importance de chaque agent
- Le début de chaque session.



Une fois les contraintes rentrées, les Agenda1 envoient au Repartisseur un message pour dire que leurs contraintes sont entrées. Une fois les contraintes de tous les agents rentrées, le Repartisseur envoie un message à tous les agents pour leur dire de commencer le calcul.

Dans cette version, le calcul lui-même a été modifié. En effet, maintenant, s'il n'y a pas de jour correspondant à un rendez-vous pour tous les agents, un ou plusieurs agents sont bannis. Le choix des agents à bannir s'effectue en choisissant tous les agents qui ne peuvent pas venir sur une période, la période étant choisie en fonction du nombre d'agents et de leur importance.

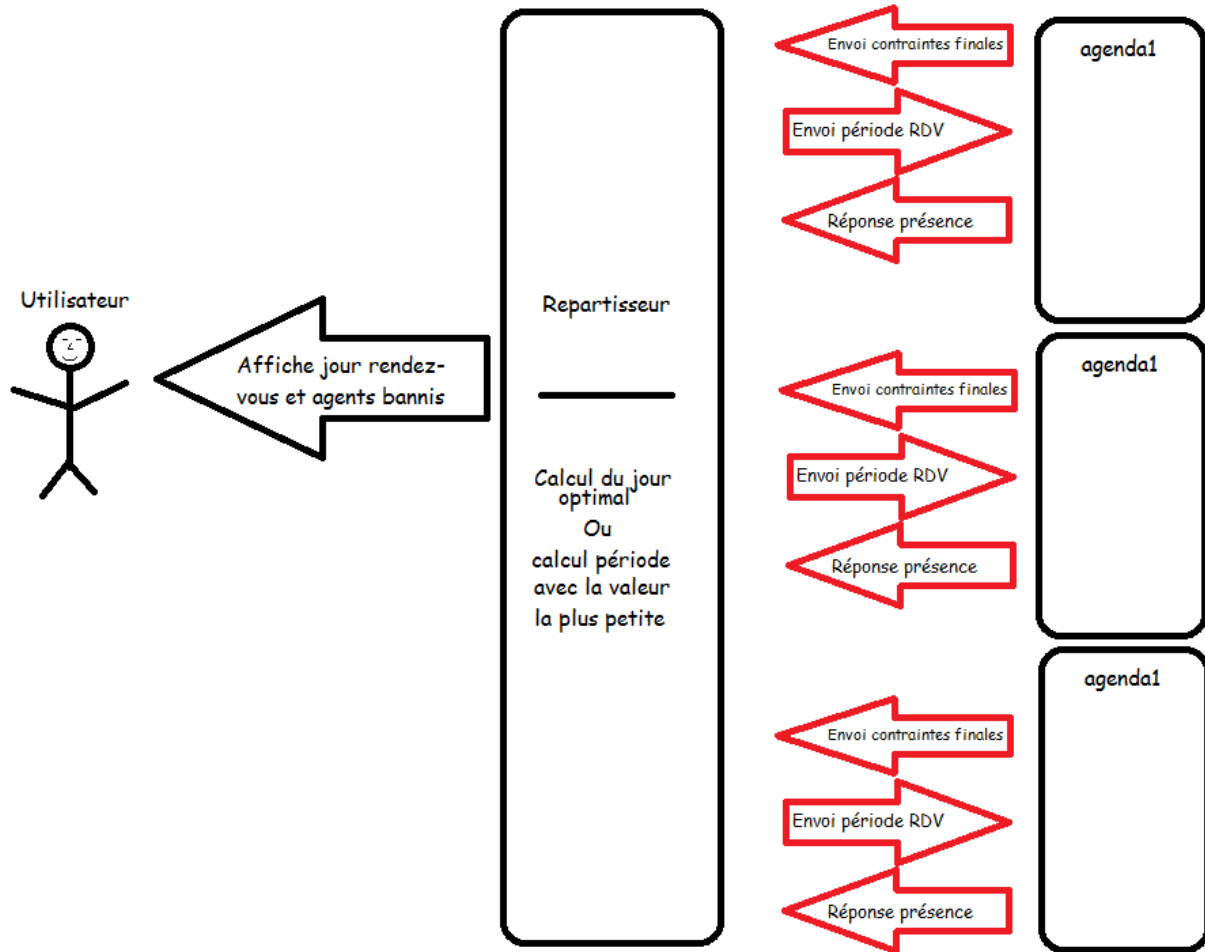
Le calcul pour remplir le tableau des contraintes finales s'effectue de la même manière que dans la première version.

Ensuite, une fois les contraintes reçues, le Repartisseur vérifie si une solution existe.

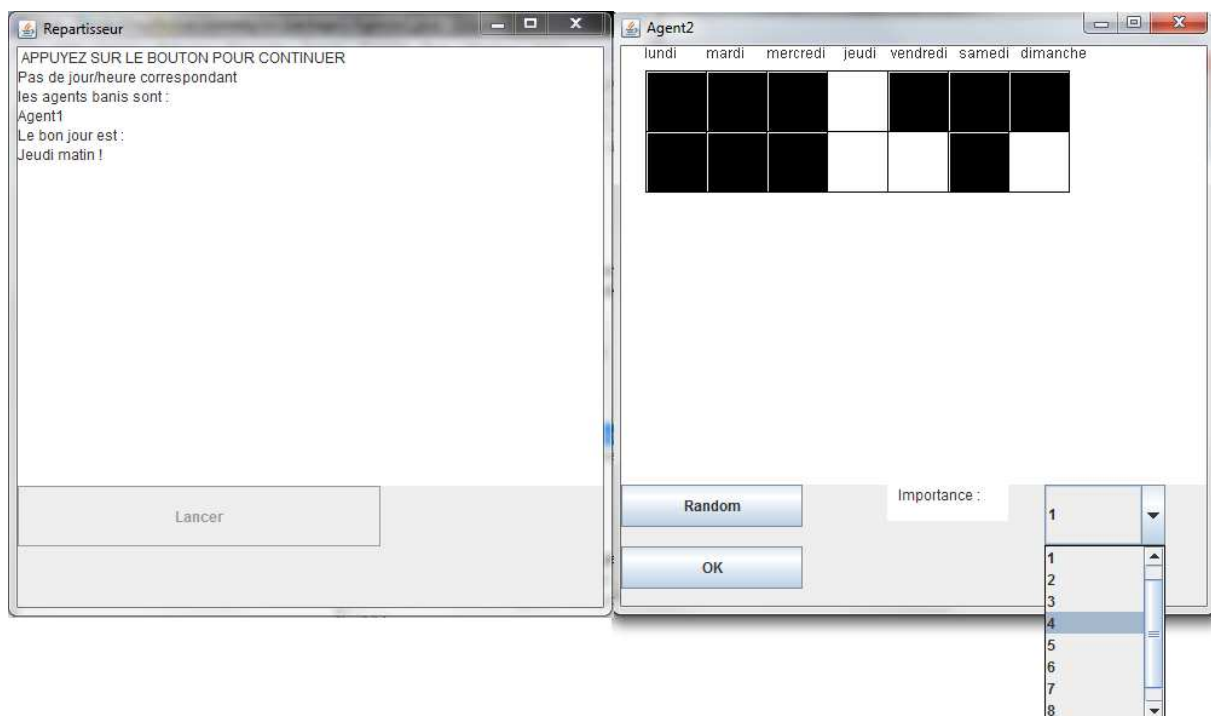
Si ce n'est pas le cas, il envoie aux agents la période sur laquelle la valeur (nombre d'agents ayant cette période indisponible multiplié par leur importance) la plus petite.

Les agents regardent ensuite s'ils étaient absents sur cette période et envoient la réponse au Repartisseur.

Le Repartisseur montre ensuite à l'utilisateur les agents bannis et la date de rendez-vous.



Le second changement important de cette version est que les données rentrées par l'utilisateur se font via une interface graphique cliquable.



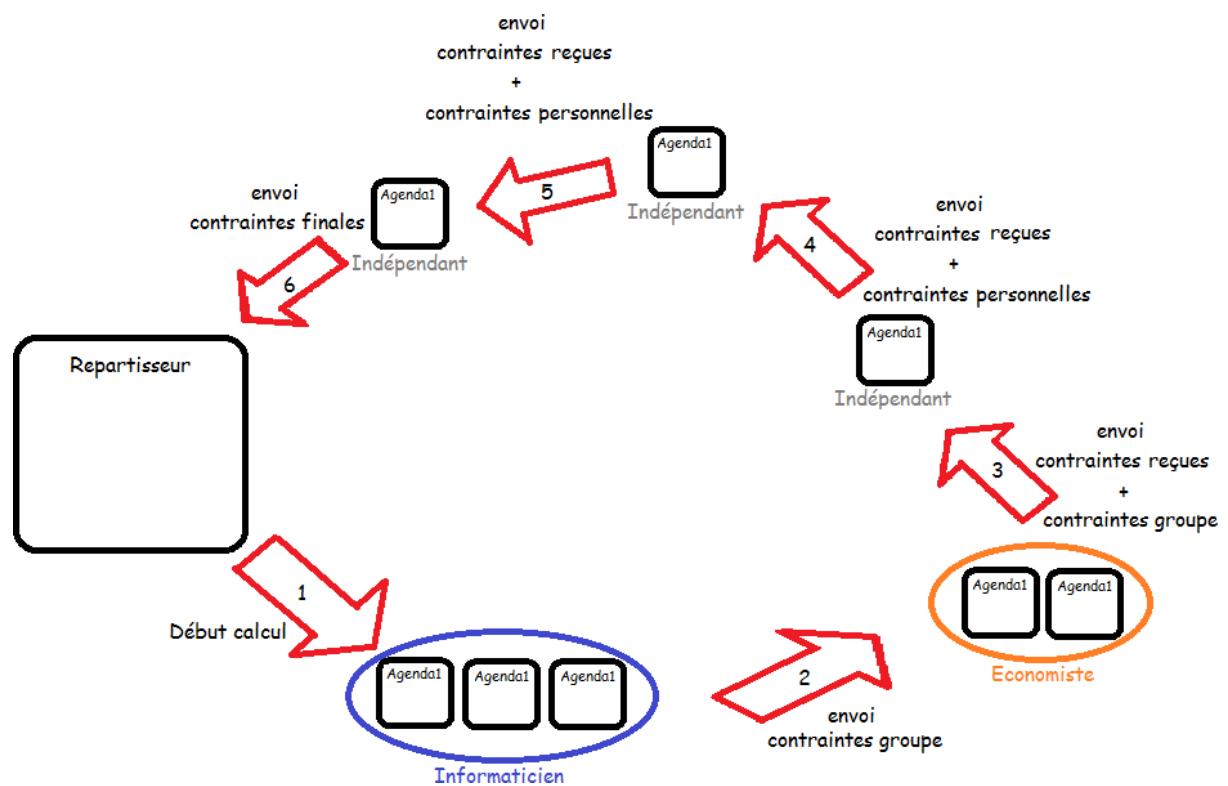
III.3. Analyse version finale

Dans la version finale nous voulions que le programme corresponde plus à un programme multi-agents. Nous avons donc instauré un système de rôles prédéfinis auquel peut appartenir chaque agent.

L'utilisateur choisit à quel rôle chaque agent appartient. Initialement, chaque agent est indépendant. Lorsque plusieurs agents appartiennent à un groupe, ils ont des contraintes de groupe (somme des contraintes personnelles de chaque agent du groupe).

Avant de commencer le calcul, chaque groupe crée un tableau de groupe, ce sera ce tableau et non celui personnel qui sera pris en compte pour le calcul des contraintes finales envoyé au Repartisseur. Seul les Indépendants utiliseront leurs contraintes personnelles.

Le calcul des contraintes finale a donc été largement revu et optimisé car le nombre de message pour ce calcul a été divisé par dix.



Pour créer les contraintes finales le Repartisseur envoie un message à un groupe aléatoirement lui disant de commencer la création du tableau des contraintes finales.

Le groupe qui le reçoit crée un tableau de contraintes et y insère ses contraintes de groupe. Il l'envoie ensuite à un autre groupe (si il existe) qui n'a pas encore rentré ses contraintes.

Cet autre groupe ajoutera à ce tableau de contraintes finales ses propres contraintes de groupe et l'enverra à son tour à un autre groupe (toujours si il existe) qui n'a pas encore rentré ses contraintes.

Lorsqu'il n'existe plus de groupe n'ayant pas rentré ses contraintes, le tableau est envoyé à un agent indépendant (encore s'il existe).

L'agent indépendant rajoute ses contraintes au tableau et l'envoie à un autre agent indépendant n'ayant pas encore ajouté ses contraintes au tableau.

Si il n'existe plus d'agents ou groupe n'ayant pas rentré ses contraintes, le dernier agent enverra le tableau de contraintes finales au Repartisseur.

Nous avons également mis en place le fait que si un agent d'un groupe est banni alors tous les agents de ce groupe seront perturbés.

Ceci apparait dans les tableaux de contraintes de groupe. Les valeurs des périodes sont augmentées proportionnellement au nombre d'agents perturbé et de la moyenne des importances des agents du groupe.

Ainsi, chaque période du tableau de contrainte de groupe ayant au moins un agent absent aura sa valeur augmentée de :

$$\frac{\text{moyenne d'importance des agents du groupe} \times (\text{nombre d'agents du groupe} - 1)}{2}$$

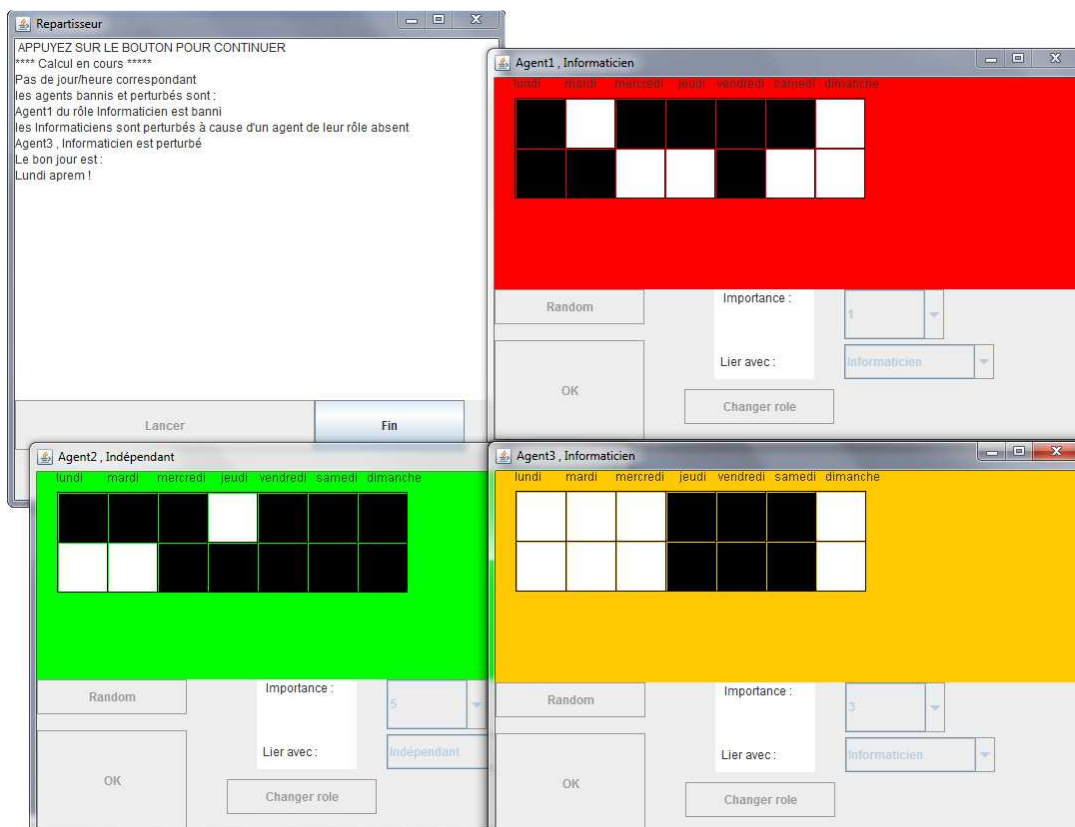
Le nombre d'agents du groupe -1 correspond aux nombre d'agents perturbés en cas de bannissement.

Le /2 pour simuler le problème lié à la perturbation. Plus on augmente ce chiffre, moins la perturbation est importante. Dans ce cas la, un agent perturbé est « à moitié présent ».

Des changements ont été également effectués sur l'interface utilisateur. Le but était que le programme soit clair et facile d'utilisation.

En effet, maintenant l'utilisateur peut regarder prendre tout le temps nécessaire pour observer ce qui s'est passé et voir le résultat. Un système de couleurs lui permet de savoir quels sont les agents absents, perturbés et présents.

Une liste déroulante accompagnée d'un bouton permet à l'utilisateur de changer, autant de fois qu'il le souhaite, le rôle de chacun des agents.

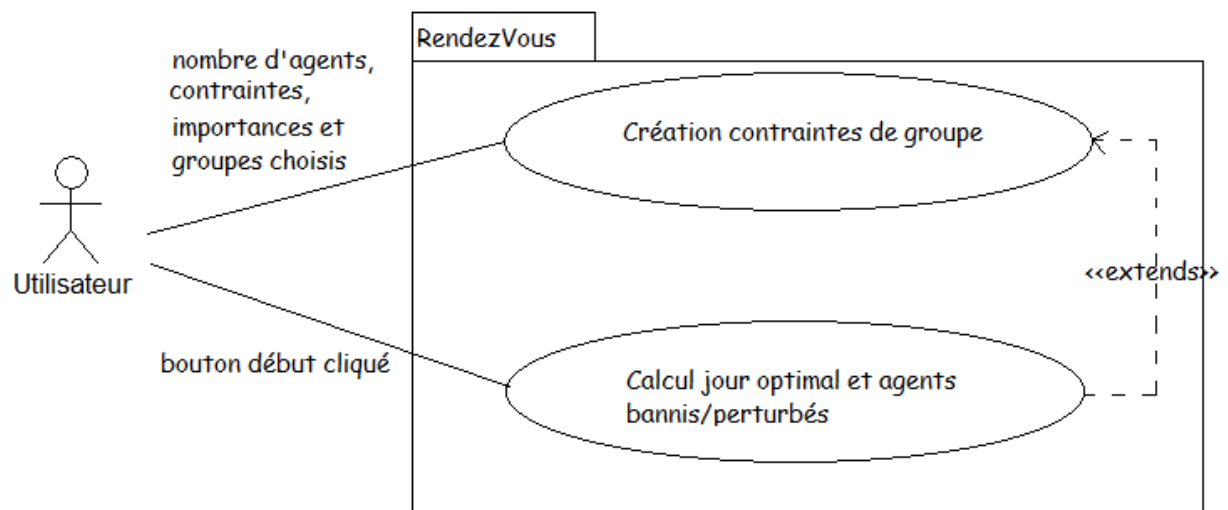


III.4. Schémas UML de la version finale

III.4.1. Diagramme des classes persistantes



III.4.2. Diagramme de cas d'utilisation (use case)



IV. Description du code source

Dans cette partie sera montré et expliqué comment sont programmées les fonctionnalités de la version finale.

IV.1. Interface

IV.1.1. Interface Repartisseur

Voici la classe Fenetre qu'utilise le Repartisseur pour afficher une fenêtre qui indique à l'utilisateur la progression du calcul et le résultat.

```
public class Fenetre extends JFrame implements ActionListener{

    boolean go=false; // boolean qui lance le répartisseur si les conditions sont réunies

    JTextArea sortie=new JTextArea(100,1000) ; // le texte

    JButton lancement=new JButton ("Lancer"); // un bouton pour lancer
    JButton fin=new JButton ("Fin"); // un bouton pour finir ou recommencer

    Repartisseur repartisseur; // pour lancer la lafin du répartisseur si on appui sur bouton fin

    public Fenetre( Repartisseur repartisseur)
    {
        this.repartisseur=repartisseur;

        setSize(500,500); // taille fenetre
        setTitle("Repartisseur"); // son titre
        JLabel placebouton = new JLabel(""); // un jlabel pour placer le bouton
        placebouton.setPreferredSize(new Dimension(300, 100)); // changement taille label
        getContentPane().add( placebouton, BorderLayout.SOUTH); // parametres pour placer un bouton

        lancement.setEnabled(false); // bouton pas cliquable au départ
        fin.setEnabled(false); // bouton pas cliquable au départ

        lancement.setBounds(0, 0, 300, 50); // changement taille boutons
        fin.setBounds(300, 0, 150, 50);

        add(sortie); // ajout du texte à la fenêtre
        placebouton.add(lancement); // ajout du bouton lancement
        placebouton.add(fin); // ajout du bouton fin

        setVisible(true);

        lancement.addActionListener(this); // place un écouteur sur le bouton lancement
        fin.addActionListener(this); // place un écouteur sur le bouton fin
    }

    La fenêtre est modifiée en fonction des évènements, ceci permet de commencer le calcul ou
    de relancer une session.
    public void attente(boolean a) // changer le fait que le bouton soit cliquable ou pas
    {
        lancement.setEnabled(a);
    }

    public void actionPerformed(ActionEvent arg0) {
        if(arg0.getSource() == lancement)// si le bouton lancer est appuyé , go (le boolean) est modifié
        {
            this.go=true;
        }
        else // si on appui sur terminer
        {
            repartisseur.recommencer();
        }
    }
}
```


IV.1.2. Interface des agents de contraintes

Voici la classe FenetreAgent permettant à chaque agent d'avoir une fenêtre pour que l'utilisateur puisse entrer les données et voir l'état de chacun des agents.

Ci-dessous sont présentées les variables de cette classe.

```
public class FenetreAgent extends JFrame implements ActionListener, WindowListener{

    //----- Variables -----
    boolean ok=false; // indique aux agents si les contraintes ont été rentrées
    JButton lancement=new JButton ("OK"); // un bouton pour valider
    JButton random= new JButton ("Random"); // Un bouton pour rentrer des contraintes aléatoires
    JButton lier= new JButton ("Changer role"); // Un bouton pour lancer la liaison
    private int[][] grille=new int[7][2]; // la grille des contraintes
    Panneau pan; // Le panneau où l'ont clique pour rentrer les contraintes
    JComboBox imp; // liste déroulante des importances des agents
    JComboBox liaison; //liste déroulante des liaisons
    Vector<String> liai=new Vector<String>(); // vecteur des liaisons choisissables dans la liste déroulante
    Vector<Integer> importances=new Vector<Integer>(); // vecteur des importances choisissables dans la liste déroulante
    private int impor; // permet de voir l'importance choisit
    Agendal agendal; // pour receptionner l'agent qui utilise cette fenêtre
    String[] nomtot; // tout les roles possibles
    String nomalier; // nom de l'agent a lier avec
    boolean quitter; //boolean pour autoriser la fenêtre à être fermée par l'utilisateur
    //-----
```

Voici l'initialisation de la partie graphique de la fenêtre. Ceci comprend les boutons, les listes déroulantes et le panneau permettant de rentrer les contraintes.

```
public FenetreAgent(String nom, Agendal agendal) // constructeur
{

    quitter=false;
    // reception tableau de noms
    this.nomtoto=new String[]{"Informaticien","Economiste","Ressources Humaines","Electricien","Cuisinier"};
    ok=false; // au début les contraintes n'ont pas été rentrées

    impor=1; // initialisation de l'importance a 1 dans la liste déroulante

    setTitle(nom); // met le nom de l'agent en titre de fenêtre
    pan=new Panneau(); // création de la zone cli

    this.agendal=agendal; // pour tuer l'agent si l'utilisateur ferme la fenêtre
    setSize(600,400); // taille fenetre

    //----- Rentrer les importances cliquables de 0 à 10 dans la liste déroulante -----
    for(int i=0;i<10;i++)
    {
        importances.add(i);
    }
    imp=new JComboBox(importances);
    //-----

    //----- Rentrer les liaison cliquables des agents dans la liste déroulante -----
    liai.add("Indépendant");
    for(int i=0;i<nomtoto.length;i++)
    {
        liai.add(nomtoto[i]);
    }
    liaison=new JComboBox(liai);
    liaison.setSelectedIndex(0);

    nomalier="Indépendant";
    //-----

    JTextArea text=new JTextArea(); // création zone de texte
    JLabel placebouton = new JLabel(""); // creation d'un label pour placer le bouton
    placebouton.setPreferredSize(new Dimension(300, 150)); // choix de sa taille
    getContentPane().add( placebouton, BorderLayout.SOUTH); // parametres pour placer un bouton

    lancement.setEnabled(true); // bouton pour lancer cliquable au début
    lier.setEnabled(true); // bouton pour lier cliquable au début
```

Voici la fonction qui permet d'effectuer les opération demandées par l'utilisateur à l'aide des boutons et listes déroulantes.

```
public void actionPerformed(ActionEvent arg0) { // écouteur des événements de la fenêtre

    int rand=0;

    if(arg0.getSource() == liaison) // si on choisi un agent à lier dans la liste déroulante
    {
        JComboBox bo=(JComboBox) arg0.getSource();

        nomalier=(String) bo.getSelectedItem();
    }
    else{
        if(arg0.getSource() == random) // si on appui sur le bouton pour mettre contraintes au random
        {
            for(int i=0;i<7;i++)
            {for(int j=0;j<2;j++)
            {
                // rentre les contraintes aléatoirement
                if(Math.random()>0.5)pan.grille[i][j]=0; else pan.grille[i][j]=1;
            }
            }
            pan.repaint(); // repaint le tableau
        }
        else{
            if(arg0.getSource() ==lier){ // si on appui sur le bouton lier
                lier();
            }
            else{
                if(arg0.getSource() == lancement) // si on appui sur le bouton pour accepter
                {
                    // on ne peut plus rien faire sur la fenêtre
                    random.setEnabled(false);liaison.setEnabled(false);lier.setEnabled(false);
                    lancement.setEnabled(false);imp.setEnabled(false);

                    pan.passageafalse();ok=true; // l'agent est débloqué
                }
                else // si on change la liste déroulante
                {
                    JComboBox bo=(JComboBox) arg0.getSource();
                    impor=(Integer) bo.getSelectedItem(); // on modifie l'importance
                }
            }
        }
    }
}
```

Voici les fonctions complémentaires de cette classe indispensables au bon fonctionnement de la fenêtre.

```
public int[][] grille() // remplit la grille de la fenêtre avec la grille de son panneau
{
    for(int i=0;i<7;i++)
    {for(int j=0;j<2;j++)
    {
        this.grille[i][j]= pan.grille()[i][j];
    }
    }
    return grille;
}

public void windowClosing(WindowEvent e) { // si l'on ferme la fenêtre ,elle est détruite et l'agent tué
    if(lancement.isEnabled() || quitter){
        agenda1.mortClic();
        agenda1.killAgent(agenda1,0);
        fermer();
    }
}
```

Voici la classe Panneau qui représente l'interface cliquable pour le choix des contraintes de chaque agent.

Ci-dessous, son constructeur et ses fonctions de base.

```
public class Panneau extends JPanel implements MouseListener
{
    public int[][] grille=new int[7][2] ; // grilles des contraintes rentrés par l'utilisateur
    boolean ok; // sert a savoir si on peut cliquer sur la grille de choix

    public Panneau() // constructeur
    {
        ok=true; // on peut au début cliquer sur la grille de choix
        setBackground(Color.WHITE);
        addMouseListener(this); // écouteur sur la souris
        setVisible(true);
    }

    public int[][] grille() // la fenêtre utilise cette methode pour recevoir la grille du panneau
    {
        return grille;
    }

    public void passageafalse() // pour rendre la grille non cliquable
    {
        ok=false;
    }

    public void changeCouleur(String coul)
    {
        if(coul.equals("bon")) setBackground(Color.green);
        else if(coul.equals("mauvais")) setBackground(Color.red);
        else if (coul.equals("groupemauvais")) setBackground(Color.orange);
    }
}
```

Voici la fonction qui permet de redessiner le panneau à chaque clic.

```
public void paintComponent(Graphics g) // ce que peindra le panneau a chaque clique
{
    super.paintComponent(g);
    // indique quel jour on clique
    g.drawString("    lundi      mardi      mercredi      jeudi      vendredi      samedi      dimanche",10, 10);

    for(int i=0;i<7;i++) // rempli en noir ou blanc en fonction de la valeur de la grille des contraintes
    {for(int j=0;j<2;j++)
    {
        g.setColor(Color.black);
        g.drawRect(i*50+20,j*50+20,50,50);
        switch (grille[i][j])
        {
            case 0:g.setColor(Color.white); break;
            case 1:g.setColor(Color.black); break;

            default:break;
        }
        g.fillRect(i*50+22,j*50+22,48,48); // dessine le rectangle rempli
    }
    }
}
```

Voici les fonctions qui permettent de détecter où l'utilisateur a cliqué et agir en conséquence.

```
public void mouseReleased(MouseEvent arg0) { // ce qui se passe quand on clique
    // transforme les coordonnées absolues (pixel) en zone de la grille des contraintes
    int[] casetouché = zone(arg0.getX(),arg0.getY());

    if(casetouché[0]!=-1 && casetouché[1]!=-1){ // si l'utilisateur n'as pas cliqué a coté
        //remplace la contrainte par rapport à la valeur précédente(0 si 1 ou 1 si 0 avant)
        if(ok){grille[casetouché[0]-1][casetouché[1]-1]=1-grille[casetouché[0]-1][casetouché[1]-1];}
    }
    repaint(); // repaint le panneau

    int cpt=0;int[] contrainte=new int[14];
    for(int i=0;i<7;i++)
    {for(int j=0;j<2;j++)
    {
        contrainte[cpt]=grille()[i][j]; // initialisation des contraintes
        cpt++;
    }
    }
}

public int[] zone(int x,int y) // transforme des variables de position en pixel en coordonné d'un tableau de 7 sur 2
{
    int tab[]=new int[2];
    // si l'utilisateur clique a coté le tableau sera à -1
    x=x-20;y=y-20;if(x<0 || x>350 || y<0 || y>100){tab[0]=-1;tab[1]=-1;return tab;}
    if (x>=0 && x<50)          tab[0]=1;
    if (x>=50 && x<100)         tab[0]=2;
    if (x>=100 && x<150)        tab[0]=3;
    if (x>=150 && x<200)        tab[0]=4;
    if (x>=200 && x<250)        tab[0]=5;
    if (x>=250 && x<300)        tab[0]=6;
    if (x>=300 && x<=350)       tab[0]=7;

    if (y>=0 && y<50)           tab[1]=1;
    if (y>=50 && y<=100)        tab[1]=2;

    return tab;
}
```

IV.2. Création du tableau de contraintes

IV.2.1. Initialisation tableau de contraintes personnelles

Voici la classe Agendal et ses variables qui seront initialisées.

```
public class Agendal extends Agent{

    //----- variables -----

    // la communauté/role/group de base de ces agents
    String myCommunity="Communauté";
    String myGroup="groupe1";
    String myRole="Indépendant";
    String nom; // identificateur de l'agent (titre de sa fenêtre

    int importance=1; // importance de l'agent

    boolean alive = true; // permet de terminer un agent

    int contrainte[]=new int[14]; // tableau des contraintes 1 ou plus =absent , 0=disponible
    int contrainteRole[]=new int[14]; //tablea des contraintes du Role en entier
    int contrainteFin[]=new int[14]; // tableau des contraintes finales
    /*
    Ces tableaux représentent les matins de chaque journée pour 1 semaine.
    Case 0 = matin lundi, case= 1 aprem lundi, case 2 = matin mardi , ect ...
    */
    // tableau pour savoir quels roles ont finit leur calcul
    boolean[] rolecaculé=new boolean[] {false,false,false,false,false};
    // Dans l'ordre : "Informaticien", "Economiste", "Ressources Humaines", "Electricien", "Cuisinier";

    FenetreAgent fen; // fenêtre de l'agent
}
```

L'agent à sa création crée sa fenêtre qui lui permettra de recevoir ses contraintes rentrée par l'utilisateur.

```
public void activate(){ // Ce que fait l'agent a sa création

    requestRole(myCommunity,myGroup,myRole); // chaque agent rentre dans le groupe

    //----- Reçoit le nom de l'agent attribué -----
    XMLMessage nomagen= new XMLMessage("");
    nomagen= (XMLMessage) waitNextMessage();

    nom=nomagen.toString();

    //-----

    fen=new FenetreAgent(nom+" , "+myRole,this); // création de la fenetre
}
```

Voici comment l'agent initialise son tableau de contraintes et puis envoi un message au Repartisseur pour indiquer qu'il est prêt. Il attend ensuite un message du Repartisseur qui indique que tous les agents sont prêts.

```

----- Attente de l'utilisateur puis initialisation des contraintes de l'agent -----

while(!fen.ok){//attente des contraintes rentrée par l'utilisateur
    nextMessage(); // vide sa pile de message
    pause(100); //pour ne pas fatiguer le processeur
}

int Cptcontraintes=0;
importance=fen.importance(); // recuperation de l'importance choisi

for(int i=0;i<7;i++)
{for(int j=0;j<2;j++)
{
    contrainte[Cptcontraintes]=fen.grille()[i][j]*importance;// initialisation des contraintes
    // initialisation des contraintes du Role
    if(!myRole.equals("Indépendant"))contrainteRole[Cptcontraintes]=fen.grille()[i][j]*importance;
    else contrainteRole[Cptcontraintes]=0; // si je suis indépendant mon tableau de role est vide

    contraintefin[Cptcontraintes]=fen.grille()[i][j]*importance; // initialisation des contraintes finales
    Cptcontraintes++;
}
}

//-----

//message envoyé au repartisseur pour dire que les contraintes ont bien été entrées
sendMessage(myCommunity, myGroup,"repartisseur", new XMLMessage("contraintes ok"));

// ----- Attente top départ du répartisseur -----
Message messagedebut =new Message();
while(messagedebut.toString()!="Commencez"){// au cas ou un agent se ferait tuer
messagedebut = waitNextMessage();} //attente du message du répartisseur pour lancer la recherche

fen.quitter=true;//boolean pour autoriser la fenêtre à être fermée par l'utilisateur

-----

```

IV.2.2. Création du tableau de contraintes de groupe

Voici le code permettant de remplir le tableau de contraintes de groupe. La prise en compte des agents perturbés est également présente.

```
if(!myRole.equals("Indépendant")) // si j'appartiens a un rôle particulier
{
    // j'envoi a mes membres du role mes contraintes et mon importance
    broadcastMessage(myCommunity, myGroup, myRole, new ObjectMessage(new MessageContrEtImp(contrainte,importance)));

    Message rolecontr; // pour receptionner les contraintes des autres membres du role
    int moyImp=importance; // moyenne des importances du groupe
    int nbMembreRole=0; // nombre de membre du role
    boolean sortieboucle=false; // savoir quand on sort de la boucle

    while(!sortieboucle)
    {
        rolecontr=waitNextMessage(500); // j'attend un message
        if(rolecontr==null)sortieboucle=true; // sort quand tout les membres du groupes ont envoyé un message
        if(rolecontr instanceof ObjectMessage){
            modifContrainteRole( rolecontr.clone()); // je modifie mon tableau de contraintes de role
            nbMembreRole++; // j'augmente le nombre de membre a chaque message reçu

            // on ajoute a la moyenne l'importance de l'agent qui a envoyé le message
            moyImp=moyImp+(( ObjectMessage<MessageContrEtImp>) rolecontr).getContent().importance; }
        moyImp=moyImp/(nbMembreRole+1); // on divise pour avoir la moyenne

        int ajout=(nbMembreRole*moyImp)/2;//L'ajout sur chaque contraintes est égal à
        //la moitié de la moyenne des agents du role * nombre d'agents moins un (les agents potentiellement perturbés)

        for(int i=0;i<14;i++) // on modifie le tableau du role pour prendre en compte les perturbés dans le calcul
        {
            if(contrainteRole[i]>0){
                //On ajoute l'ajout au tableau de contrainte du role pour prendre en compte le fait
                //que les agents soit perturbés en cas d'une suppression d'un agent du role
                contrainteRole[i]=contrainteRole[i]+(ajout);
            }

            // Chaque jour où le groupe a un membre d'absent a son importance augmenté de la moitié de
            //l'importance de chaque agent du role, car si on est perturbé on est 2 fois moins efficace
        }

        //j'envoi un message pour dire que je suis pret
        sendMessage(myCommunity,myGroup,"repartisseur",new XMLMessage("je suis pret"));
    }
    else //si je suis indépendant
    { // j'envoi un message pour dire que je suis pret
        sendMessage(myCommunity,myGroup,"repartisseur",new XMLMessage("je suis pret"));
    }
}

//-----
```


IV.3. Mise en commun des contraintes

Pour commencer le calcul, le Repartisseur envoie un message de départ de calcul au premier groupe existant qu'il trouve.

```
if(alive) // s'il reste des agents en vie
{
envoiRandom(new XMLMessage("depart calcul")); // fait demarrer le calcul
```

Voici la fonction qui permet cela.

```
private void envoiRandom(XMLMessage mess)
{
    if(isRole(myCommunity,myGroup,"Informaticien"))sendMessage(myCommunity,myGroup,"Informaticien", mess);
    else if(isRole(myCommunity,myGroup,"Economiste"))sendMessage(myCommunity,myGroup,"Economiste", mess);
    else if(isRole(myCommunity,myGroup,"Ressources Humaines"))sendMessage(myCommunity,myGroup,"Ressources Humaines", mess);
    else if (isRole(myCommunity,myGroup,"Electricien"))sendMessage(myCommunity,myGroup,"Electricien", mess);
    else if (isRole(myCommunity,myGroup,"Cuisinier"))sendMessage(myCommunity,myGroup,"Cuisinier", mess);
    else if(isRole(myCommunity,myGroup,"Indépendant"))sendMessage(myCommunity,myGroup,"Indépendant", mess);
}
```

Voici une fonction de la classe Agenda1 qui permet d'envoyer un tableau de contrainte à des groupes qui ne l'ont pas reçu.

Cette fonction sera utilisée lors de la création du tableau de contraintes finales.

```
private void envoiconstraintessuiv(int[] contrainteFin2, String myRole2, boolean[] rolecaculé2) {
    //fonction qui envoie le tableau de contrainte en paramètre à un rôle qui ne l'a pas déjà fait
    // et qui existe

    boolean sortiebou=false; // pour sortir de la boucle
    int parcours=0; // indice pour prendre le premier rôle disponible
    rolecaculé2[roleNum(myRole2)]=true; //on met dans le tableau que le rôle de l'agent a bien mis ses contraintes

    while(parcours<rolecaculé2.length && !sortiebou )
    {
        if(rolecaculé2[parcours]==false && isRole(myCommunity,myGroup,numRole(parcours)))
        { // si le rôle existe et qu'il a pas déjà mis ses contraintes
            sortiebou=true;
        }
        else parcours++;
    }

    if(parcours<4){ // s'il y avait un rôle restant qui n'a pas renvoyé ses contraintes
        Message(myCommunity,myGroup,numRole(parcours),new ObjectMessage(new MessageContrEtTabl(contrainteFin2,rolecaculé2)));
        // j'envoi à ce rôle le tableau de contraintes et le tableau qui indique qui a déjà rempli le tableau
    }

    // si il n'y a plus de rôles qui n'ont pas renvoyé, j'envoi à un indépendant (s'il en reste)
    if(isRole(myCommunity,myGroup,"Indépendant"))
        sendMessage(myCommunity,myGroup,"Indépendant",new ObjectMessage(contrainteFin2));
    // si tout le monde a rempli le tableau avec ses (ou celles de son groupe) contraintes, j'envoi au répartisseur
    else sendMessage(myCommunity,myGroup,"repartisseur",new ObjectMessage(contrainteFin2));
}
```


Voici le code permettant aux agents de se transférer le tableau de façon optimisée et une fois le tableau de contrainte rempli, de l'envoyer au Répartisseur.

```
//-----Calcul contraintes finales et contraintes Rôle-----

while( !fincalcul) // tant que le calcul n'est pas terminé
{

    calculcontraintes=waitNextMessage(); // on attends un message

    if(calculcontraintes.getSender().getRole()=="repartisseur") // s'il vient du répartisseur
    {
        if(calculcontraintes.toString().equals("fin calcul"))
        {fincalcul=true; } //si c'est fin calcul alors on sort de la boucle
        else if(calculcontraintes.toString().equals("depart calcul")) // si c'est départ
        {

            if(!myRole.equals("Indépendant"))
            // on envoi un message au prochain avec son premier tableau
            envoiconstraintessuiv(constrainteRole,myRole,new boolean[5]);
            else {

                leaveRole(myCommunity,myGroup,myRole); //J'abandonnemon ancien role
                // changement du role de l'agent pour ne plus recevoir le tableau
                requestRole(myCommunity,myGroup,"Indépendant finit");
                myRole="Indépendant finit";
                if(isRole(myCommunity,myGroup,"Indépendant")){ // s'il existe encore des indépendant
                    sendMessage(myCommunity,myGroup,"Indépendant", new ObjectMessage(constraintefin)); } // envoi du tableau

                else {sendMessage(myCommunity,myGroup,"repartisseur", new ObjectMessage(constraintefin));
                    } // s'il n'y a plus personne envoi au répartisseur

            }

        }

    }
    else // si le message ne vient pas du répartisseur
    {
        if(myRole!="Indépendant" && myRole!="Indépendant finit") // si j'ai un role
        {

            for(int i=0;i<constraintefin.length;i++) // j'ajoute les contraintes du groupe aux contraintes finales
            constraintefin[i]=constrainteRole[i]+((ObjectMessage<MessageContrEtTabl>) calculcontraintes).getContent().contraintes[i];

            for(int i=0;i<rolecaculé.length;i++) //je recupere ceux a qui faut pas l'envoyer
            rolecaculé[i]=((ObjectMessage<MessageContrEtTabl>) calculcontraintes).getContent().Nepasenvoi[i];

            rolecaculé[roleNum(myRole)]=true; // j'ajoute mon role a ceux qu'il ne faut pas envoyer

            envoiconstraintessuiv(constraintefin,myRole,rolecaculé); // j'envoi le message au suivant

        }
        else{ // si je suis indépendant
            for(int i=0;i<constraintefin.length;i++) // j'ajoute mes contraintes aux contraintes finales
            {constraintefin[i]=constrainte[i]+((ObjectMessage<int[]>) calculcontraintes).getContent()[i];

            }

            leaveRole(myCommunity,myGroup,myRole); //J'abandonnemon ancien role
            // changement du role de l'agent pour ne plus recevoir le tableau
            requestRole(myCommunity,myGroup,"Indépendant finit");
            myRole="Indépendant finit";
            if(isRole(myCommunity,myGroup,"Indépendant")){ // s'il existe encore des indépendant
                sendMessage(myCommunity,myGroup,"Indépendant", new ObjectMessage(constraintefin)); } // envoi du tableau

            else {sendMessage(myCommunity,myGroup,"repartisseur", new ObjectMessage(constraintefin));
                } // s'il n'y a plus personne envoi au répartisseur

        }

    }

}

//-----
```

IV.4. Prise de décision du rendez-vous

Voici le code permettant au Repartisseur de recevoir le message, envoyé par un agent, contenant le tableau de contraintes final. Le Repartisseur regarde ensuite s'il existe une solution.

```

        Message m = null; // reception messages des agents
----- Reception messages des agents -----
        while(!(m instanceof ObjectMessage)) // attente reponse finale
        {
            m = waitNextMessage(); // reception

            if (m.toString().equals("je meurs"))
            {
                // s'il n'y a plus d'agent en vie, le programme s'arrete
                f.ajoutSortieLigne(" un agent a été tué");nombreagent=nombreagent-1;
            }
            if(nombreagent<=0){alive=false;break; }
            if(m instanceof ObjectMessage){
                envoiTous(myCommunity, myGroup, new XMLMessage("fin calcul"));
                tab= (int[]) ((ObjectMessage) m).getContent(); // reception du tableau de contraintes finales
            }
        }
-----

        if(alive){ // s'il reste des agents en vie

            if(y<14){// regarde s'il y a une case du tableau des contraintes finales à 0 (tous présent)
                while( y<14 && tab[y]!=0) {

                    y++;if(y>=14)break;
                }
            }
        }
    }
}

```

IV.4.1. Il existe une solution possible

Dans le cas où il y a une solution possible, le Repartisseur affiche directement la date du rendez-vous.

```

    else { // s'il y a une solution
        // débloquent les agents qui attendent l'indice à vérifier
        envoiTous(myCommunity, myGroup, new XMLMessage("meurs"));

    }

    f.ajoutSortieLigne("Le bon jour est : ");
    ----- affichage de la date la plus rapide-----
    switch (y)
    {
        case 0: f.ajoutSortieLigne("Lundi matin ! "); break;
        case 1: f.ajoutSortieLigne("Lundi aprem ! "); break;
        case 2: f.ajoutSortieLigne("Mardi matin ! "); break;
        case 3: f.ajoutSortieLigne("Mardi aprem ! "); break;
        case 4: f.ajoutSortieLigne("Mercredi matin ! "); break;
        case 5: f.ajoutSortieLigne("Mercredi aprem ! "); break;
        case 6: f.ajoutSortieLigne("Jeudi matin ! "); break;
        case 7: f.ajoutSortieLigne("Jeudi aprem ! "); break;
        case 8: f.ajoutSortieLigne("Vendredi matin ! "); break;
        case 9: f.ajoutSortieLigne("Vendredi aprem ! "); break;
        case 10: f.ajoutSortieLigne("Samedi matin ! "); break;
        case 11: f.ajoutSortieLigne("Samedi aprem ! "); break;
        case 12: f.ajoutSortieLigne("Dimanche matin ! "); break;
        case 13: f.ajoutSortieLigne("Dimanche aprem ! "); break;

        default: break;
    }
}

```

IV.4.2. Pas de solution, bannissement d'agent(s) pour arriver à un accord

Dans le cas où il n'y a pas de solution, le Repartisseur l'indique à l'utilisateur et envoie un message aux agents contenant l'indice du jour du rendez-vous choisi à la première période ayant la valeur la plus petite.

Il affiche ensuite les agents bannis et perturbés en fonction des messages qu'il reçoit des agents.

```
if(y==14){

    f.ajoutSortieLigne("Pas de jour/heure correspondant");
    //variables pour determiner la case ayant la valeur la plus petite
    //dépendant du nombre et importance des agents à la periode
    int min=tab[0];
    int indice=0;
    for(int j=1;j<14;j++)
    {
        if(tab[j]<min){min=tab[j];indice=j;} // recupere l'indice de la valeur minimum
    }
    // envoi l'indice aux agent pour qu'ils regardent s'ils sont concernés
    envoiTous(myCommunity, myGroup, new ObjectMessage(indice));
    f.ajoutSortieLigne("les agents bannis et perturbés sont : ");
    XMLMessage agentpasbon; // message qui receptionne les agents bannis
    for(int i=0;i<nombreakagent;i++)
    {
        agentpasbon=(XMLMessage) waitNextMessage();if(agentpasbon.toString().equals("je meurs"))i--;
        if(!agentpasbon.toString().equals("pas moi") && !agentpasbon.toString().equals("je meurs")
        && !agentpasbon.toString().contains("perturbé"))
        {
            // si le message contient le nom d'un agent

            f.ajoutSortieLigne(agentpasbon.toString()+" est banni"); // indique a l'utilisateur les agents banni
            if(!agentpasbon.toString().contains("Indépendant"))
            // indique a l'utilisateur les agents perturbé
            f.ajoutSortieLigne("les "+agentpasbon.getSender().getRole()+"s sont perturbés à cause d'un agent de leur rôle absent");

        }
        else{ if(!agentpasbon.toString().equals("pas moi") && !agentpasbon.toString().equals("je meurs")){
            f.ajoutSortieLigne(agentpasbon.toString());
        }
    }
}
y=indice; // le bon jour est celui où la valeur du tableau des contrainte est la plus petite
}
```

Les agents réceptionnent le jour du rendez-vous et vérifient s'ils sont absent ou pas ce jour et si un agent de leur groupe est absent ce jour la.

Ils changent ensuite la couleur de leur fenêtre en fonction.

Ils envoient enfin un message au Repartisseur pour indiquer s'ils sont absents, présents ou perturbés.

```
//receptionne le message qui contient la position où il ne faut pas être (ex : lundi matin)
Message posiMauvais= waitNextMessage();
if (posiMauvais instanceof ObjectMessage)
{
    if (myRole.equals("Indépendant finit")){
        leaveRole(myCommunity,myGroup,myRole);//J'abandonnemon ancien role
        // changement du role de l'agent pour ne plus recevoir le tableau
        requestRole(myCommunity,myGroup,"Indépendant");
        myRole="Indépendant";
    }
}

int pos= (Integer) ((ObjectMessage<?>) posiMauvais).getContent(); // reception dans un tableau

if (contrainte[pos]>0) // si cet agent est absent ce jour la (ou son agent lié)
// envoi au répartisseur son nom pour l'indiquer a l'utilisateur
sendMessage(myCommunity, myGroup, "repartisseur", new XMLMessage(nom+" du rôle "+myRole));
fen.pan.changeCouleur("mauvais");

}
else
{
    if (contrainteRole[pos]>0)
    {
        //envoi au répartisseur pour indiquer qu'il est perturbé par l'absence d'un membre du Role
        sendMessage(myCommunity, myGroup, "repartisseur", new XMLMessage(nom+" , "+myRole+" est perturbé"));
        fen.pan.changeCouleur("groupemauvais");
    }
    else{//envoi au répartisseur pour indiquer qu'il n'est pas concerné
        sendMessage(myCommunity, myGroup, "repartisseur", new XMLMessage("pas moi"));
        fen.pan.changeCouleur("bon");
    }
}
}
}

// s'il y a une solution tout le monde est peint en vert
else if (posiMauvais.toString().equals("meurs")) fen.pan.changeCouleur("bon");
```

V. Conclusion

V.1. Bilan

Au début de notre projet, il nous a été demandé de créer une application ou d'en améliorer une afin de fournir aux nouveaux utilisateurs de MadKit des démonstrations supplémentaires de ses fonctionnalités.

Nous avons donc créé un programme construit avec des fonctionnalités de MadKit sur un exemple de la vie de tous les jours. Les nouveaux utilisateurs de MadKit peuvent donc y voir l'utilité de la programmation multi-agent et les différentes possibilités qu'offre cette plateforme.

Une fois le thème de notre logiciel adopté, nous nous sommes fixés des objectifs de programmation. Notre logiciel devait être simple d'utilisation, les résultats du calcul devaient être clairs et compréhensibles, le programme devait utiliser des fonctions de MadKit et le code devait être très commenté pour permettre à l'utilisateur de comprendre le code.

Au final, notre logiciel comporte peu de bouton. Ces boutons sont volumineux et nommés par rapport à leur fonction. De plus, à chaque étape, seuls les boutons à utiliser sont disponible afin de guider l'utilisateur pas à pas vers le résultat qu'il souhaite. Le résultat est également affiché clairement via un code de couleur qui permet voir rapidement le résultat.

Dans le code du logiciel, différentes fonctions de MadKit sont utilisées notamment pour la gestion des groupes et l'envoi de messages. De plus, le code a été divisé en plusieurs parties distinctes et commentés à chaque étape.

A la base nous nous étions fixés la réalisation de plusieurs logiciels mais étant donné les difficultés rencontrées pendant la création du projet RendezVous et de l'ampleur que représentait ce projet, nous avons décidé de nous concentrer sur ce logiciel en y ajoutant de nouvelles fonctionnalités.

V.2. Ce que ce projet nous a apporté

La réalisation de ce projet nous a beaucoup apporté. Tout d'abord, elle nous a appris ce qu'est le travail en groupe sur une longue durée. Entre autre, nous avons appris à :

- répartir les tâches entre deux personnes
- sécuriser les données pour éviter toute perte
- mettre en commun le code créé individuellement

De plus, au niveau informatique nous avons découvert et maîtrisé de nouvelles techniques de programmation (multi-agents) et perfectionné notre programmation Java (notamment le graphisme).

Enfin, ça nous a obligé à rédiger un rapport conséquent, structuré et destinés à des lecteurs appartenant à différents domaines. Ceci nous aura été bénéfique pour des projets futurs que ce soit dans les études ou dans la vie professionnelle.

V.3.Améliorations possibles du logiciel

Pour que le logiciel ait un réel intérêt en lui-même (sans le côté pédagogique), il aurait fallu que le logiciel soit en réseaux, c'est-à-dire que chaque agent soit une personne qui rentre ses contraintes et que la réponse soit donnée à tous les utilisateurs. De plus, le choix de périodes plus précises (pas seulement demi-journées) aurait pu être intéressant.

Si nous devions refaire ce projet, nous déciderions de discuter plus tôt avec nos collègues qui travaillaient sur un sujet similaire afin de gagner du temps sur des problèmes déjà résolus.

VI. Bibliographie et logiciels utilisés

VI.1. Bibliographie

<http://www.madkit.org/>: Site officiel de MadKit. Il nous a permis de télécharger les différentes versions de MadKit et de communiquer avec notre tuteur via le forum.

<http://www.siteduzero.com/> : Site qui nous a permis de résoudre des problèmes liés à la programmation java en général.

La MadKitDoc : Documentation de la librairie MadKit. Elle nous a permis de connaître les différentes méthodes et classes de MadKit.

VI.2. logiciels utilisés

Eclipse : Environnement utilisé pour programmer en Java.

MadKit version 4.2.0: Librairie utilisée pour tester les fonctionnalités de MadKit.

MadKit version 5.0: Librairie utilisée pour programmer le logiciel.

Boulim : Logiciel utilisé pour faire et utiliser des représentations UML.

VII. Annexes

VII.1. Liens utiles

Où télécharger notre logiciel : <http://www.madkit.org/démonstration/RendezVous/>

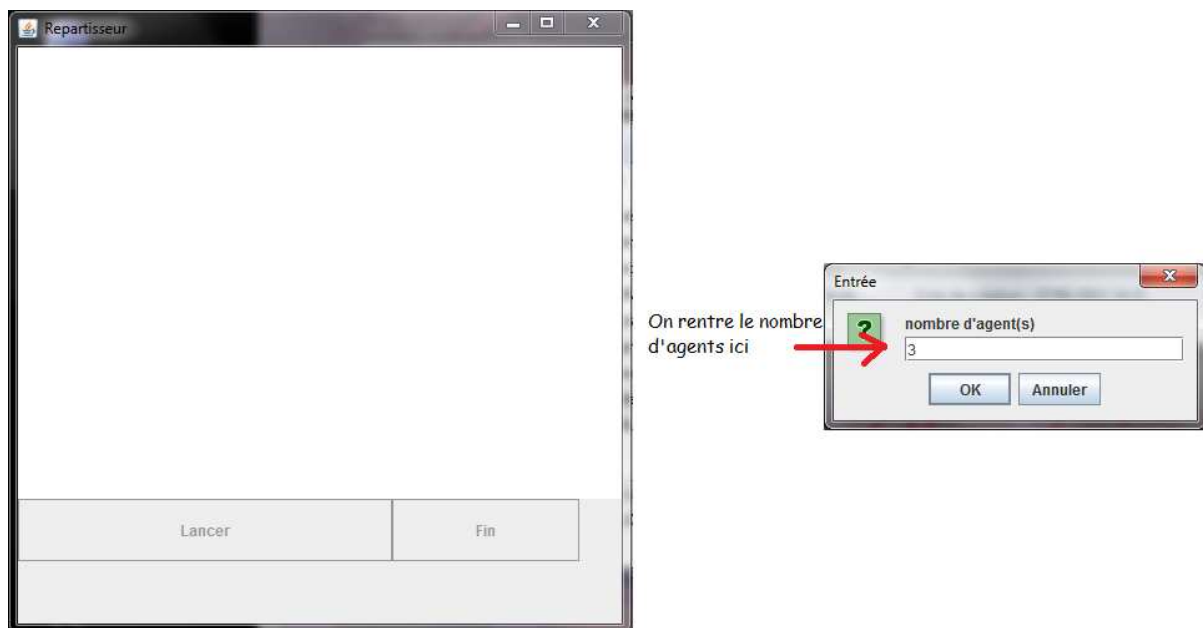
Blog suivant l'avancement de nos travaux : <http://mauringinerprojet.blogspot.com/>

Où télécharger MadKit : <http://www.madkit.org/>

VII.2. Exemple d'utilisation

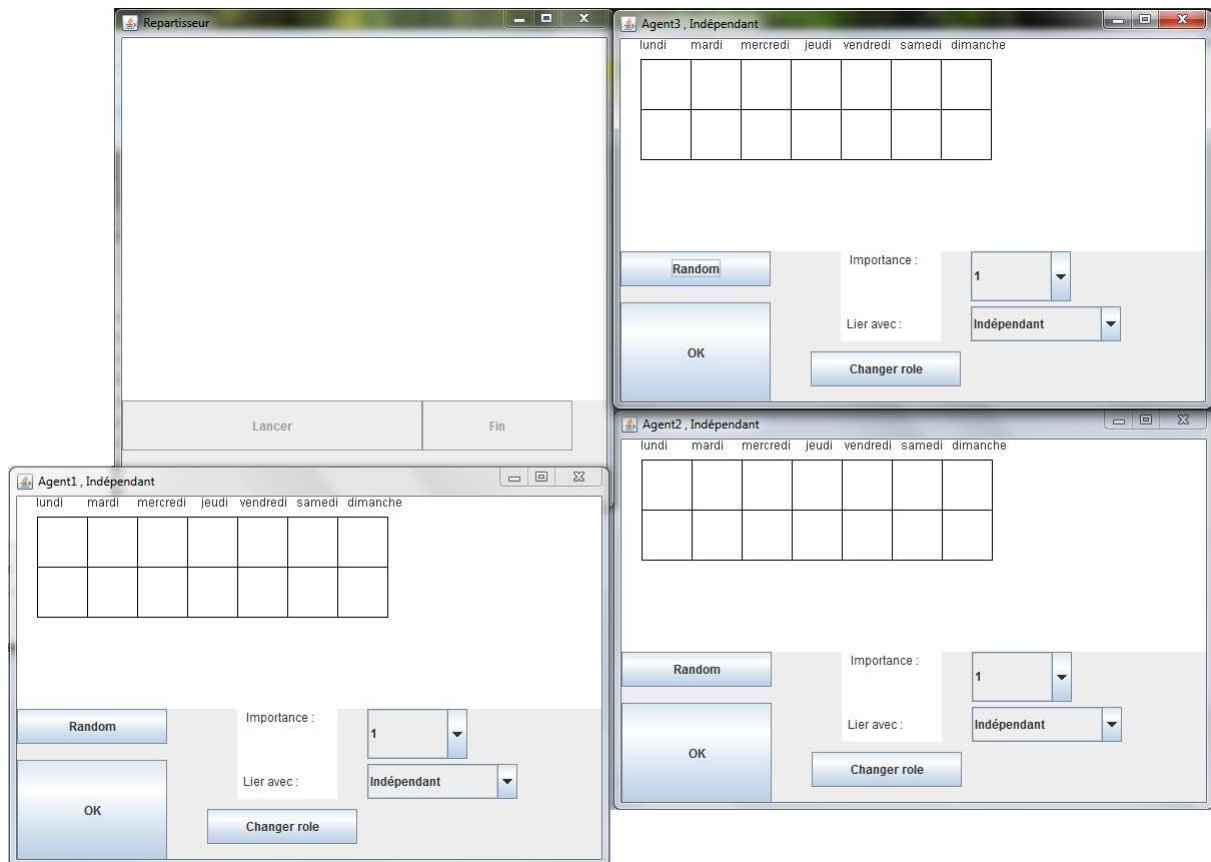
Nous allons ici vous montrer point par point un scénario d'utilisation simple avec des explications.

Etape 1 :

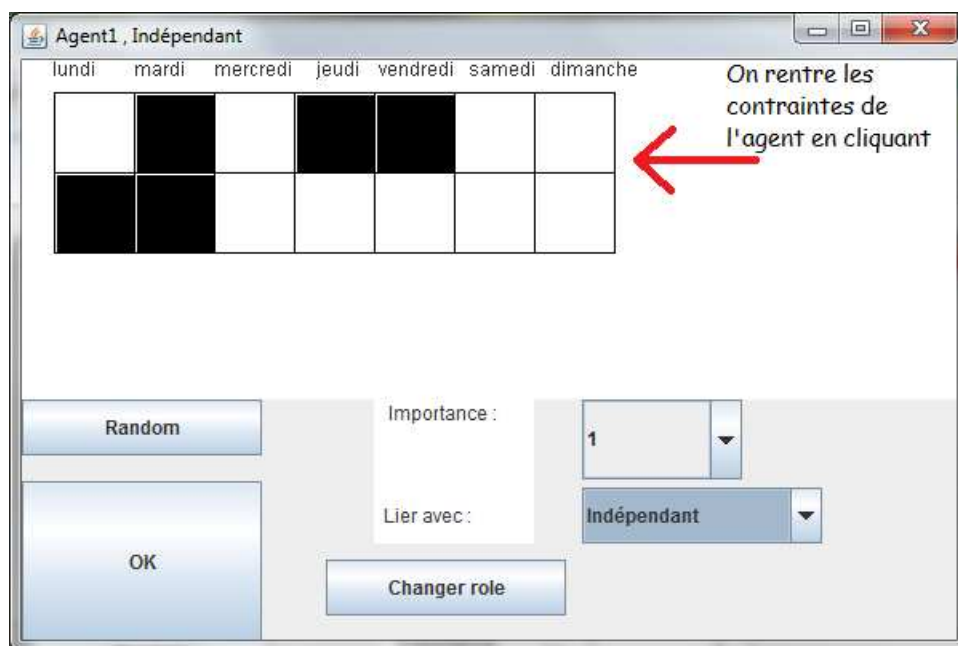


Etape 2 :

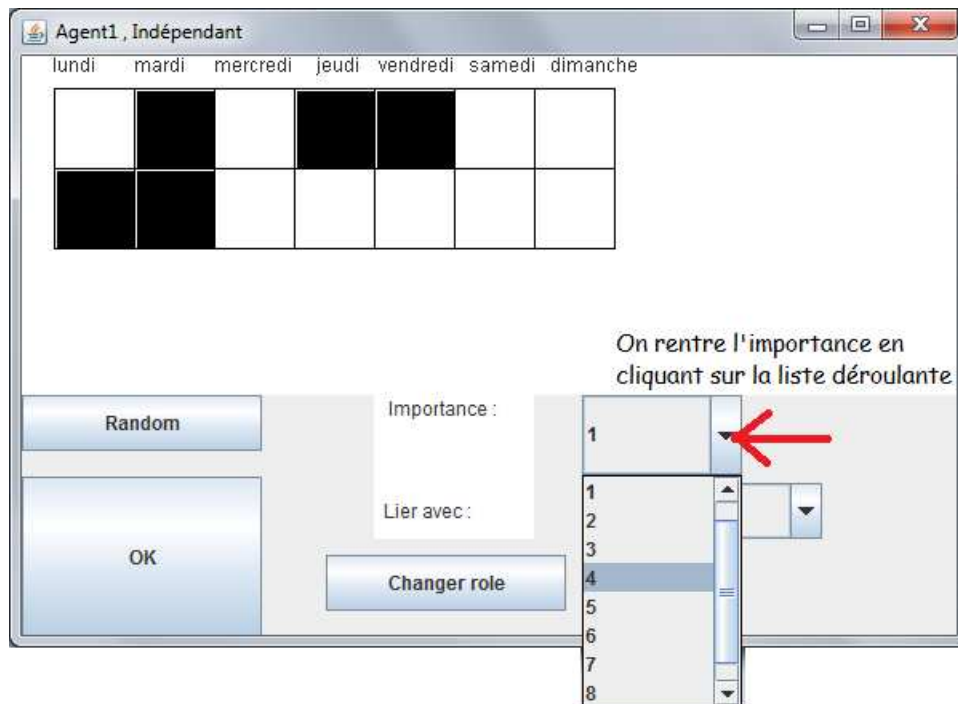
Les agents se créent et attendent l'entrée de données.



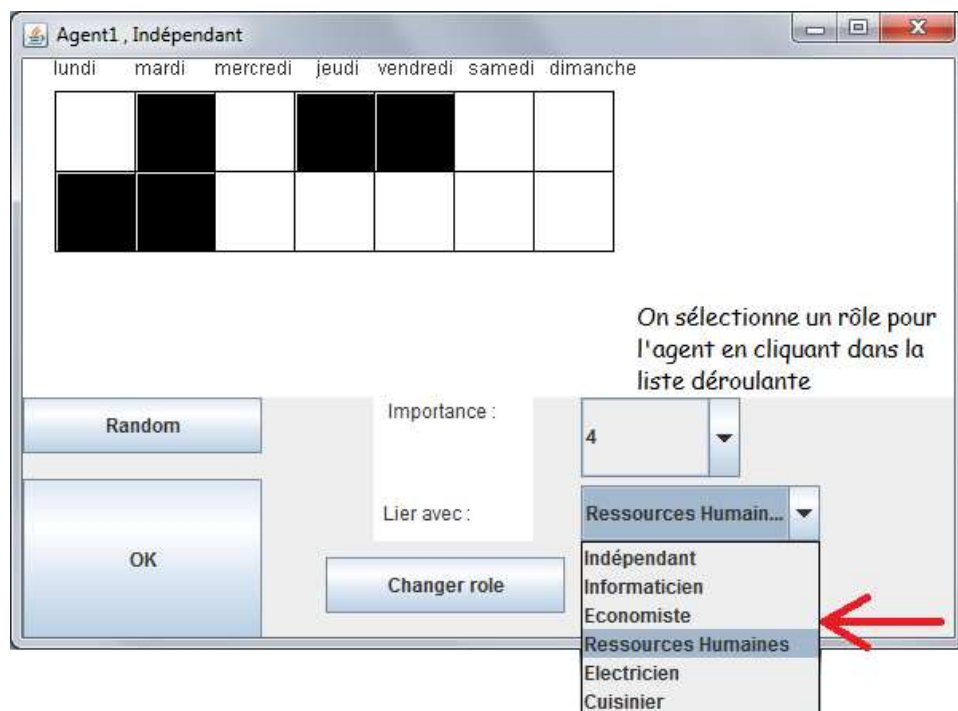
Etape 3 :



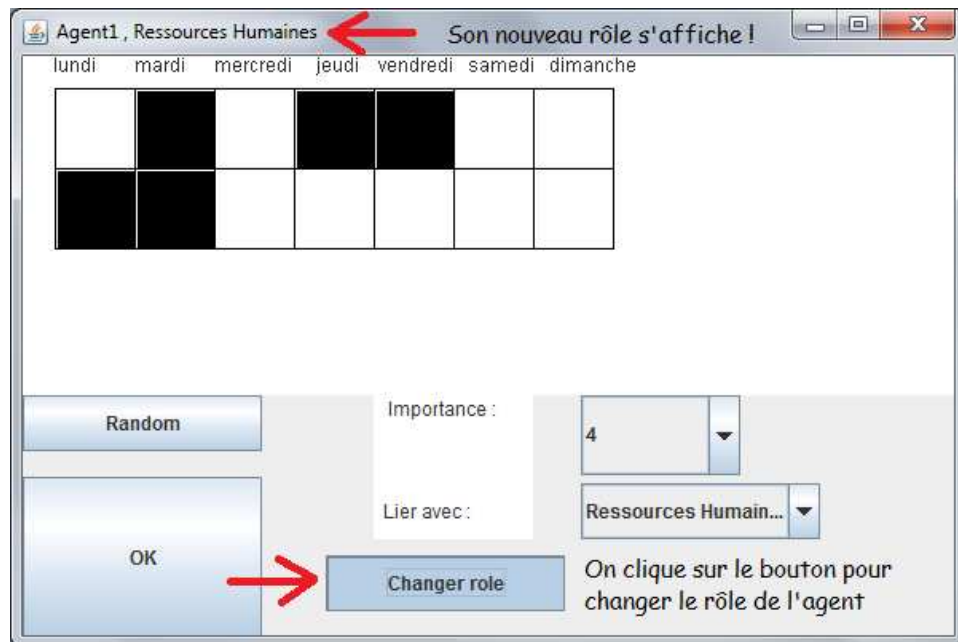
Etape 4 :



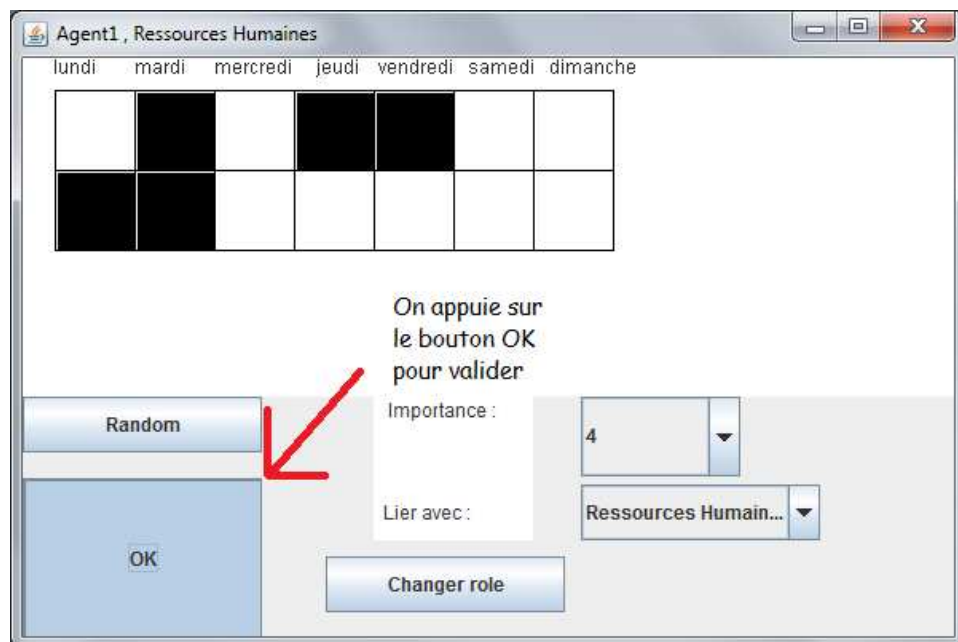
Etape 5 :



Etape 6 :

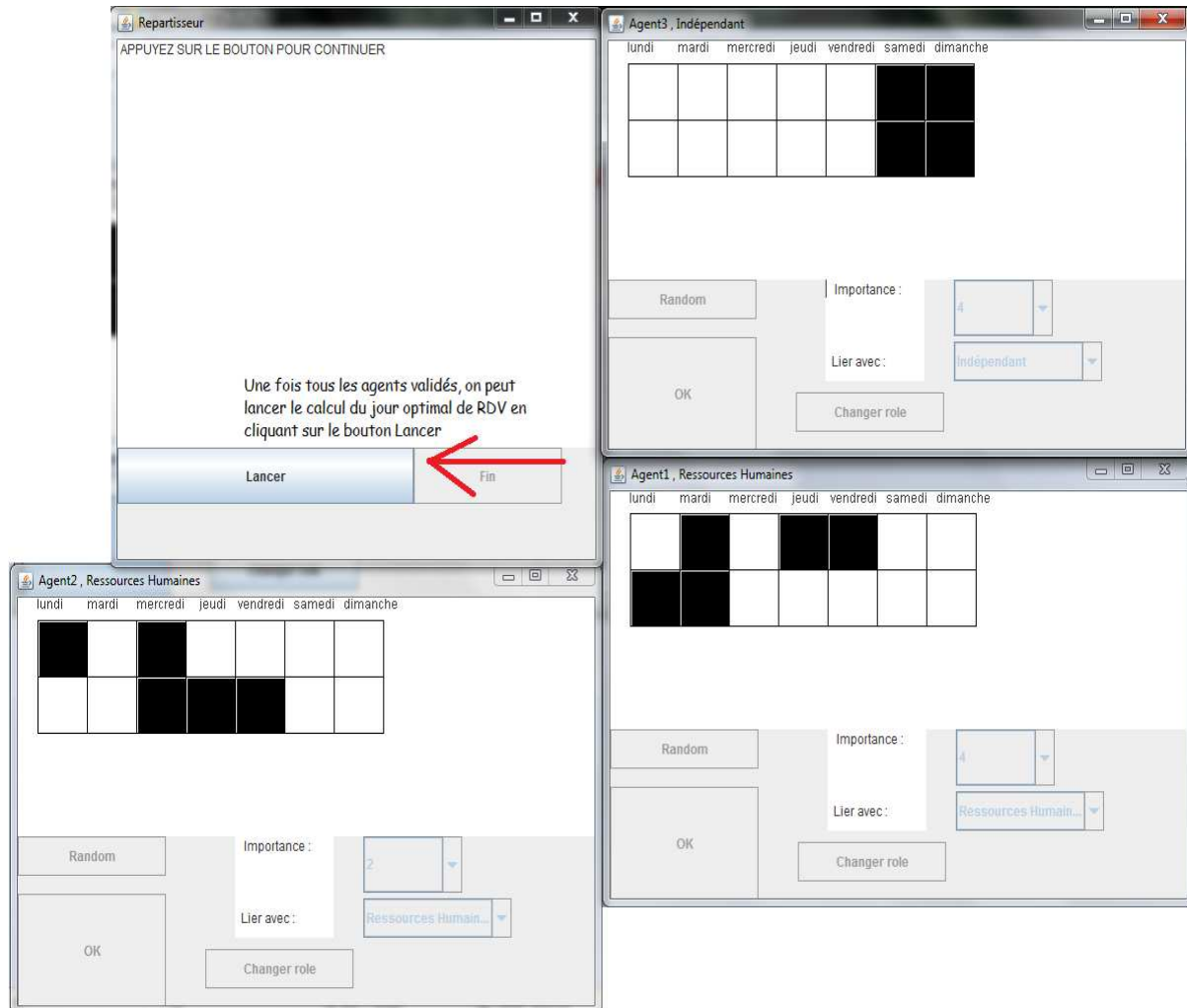


Etape 7 :

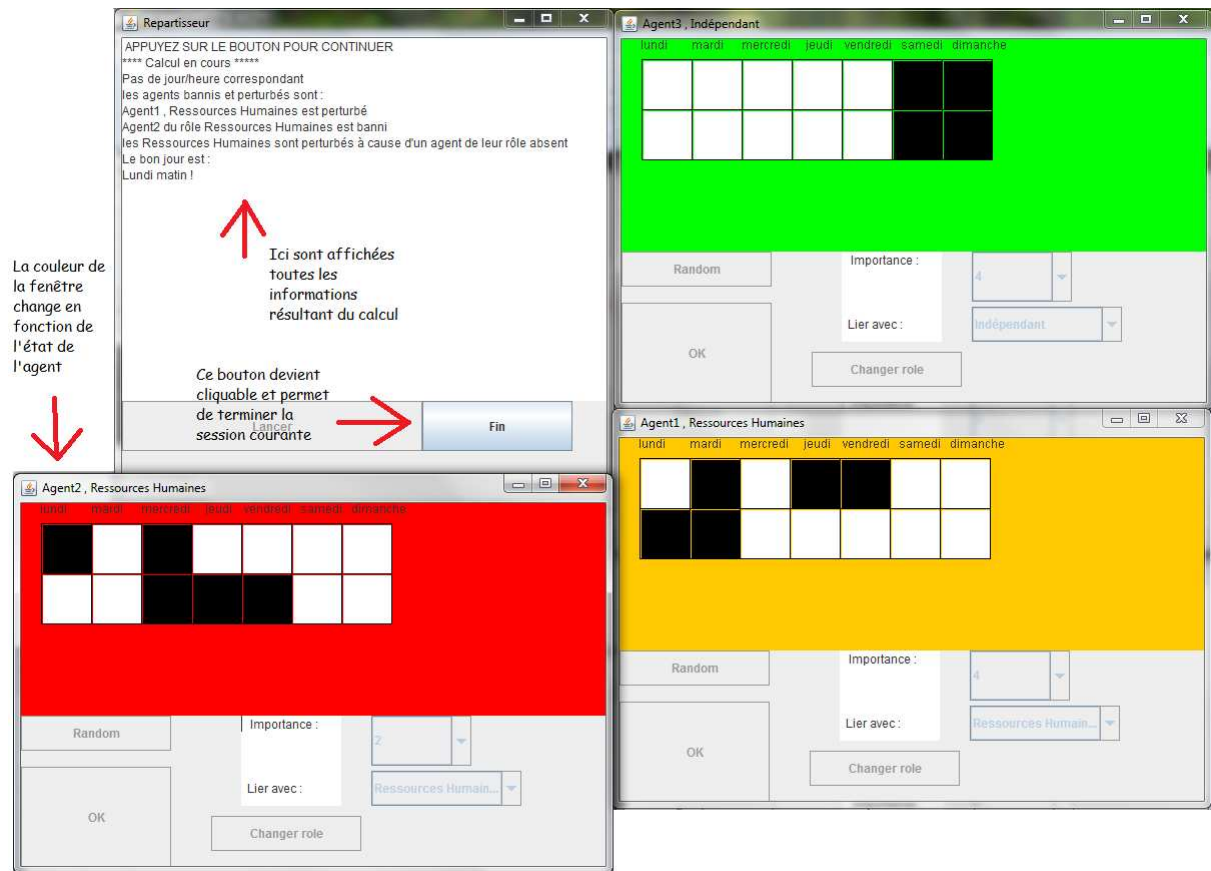


Etape 8 :

Le programme restera en attente tant que l'utilisateur n'aura pas donné son feu vert en cliquant sur le bouton Lancer.



Etape 9 :



La couleur de la fenêtre change en fonction de l'état de l'agent

↑ Ici sont affichées toutes les informations résultant du calcul

↓ Ce bouton devient cliquable et permet de terminer la session courante

Repartisseur

APPUYEZ SUR LE BOUTON POUR CONTINUER

**** Calcul en cours ****

Pas de jour/heure correspondant

les agents bannis et perturbés sont :

Agent1, Ressources Humaines est perturbé

Agent2 du rôle Ressources Humaines est banni

les Ressources Humaines sont perturbés à cause d'un agent de leur rôle absent

Le bon jour est :

Lundi matin !

Agent3, Indépendant

lundi	mardi	mercredi	jeudi	vendredi	samedi	dimanche

Random Importance : 4

OK Lier avec : Indépendant

Changer role

Agent2, Ressources Humaines

lundi	mardi	mercredi	jeudi	vendredi	samedi	dimanche

Random Importance : 2

OK Lier avec : Ressources Humain...

Changer role

Agent1, Ressources Humaines

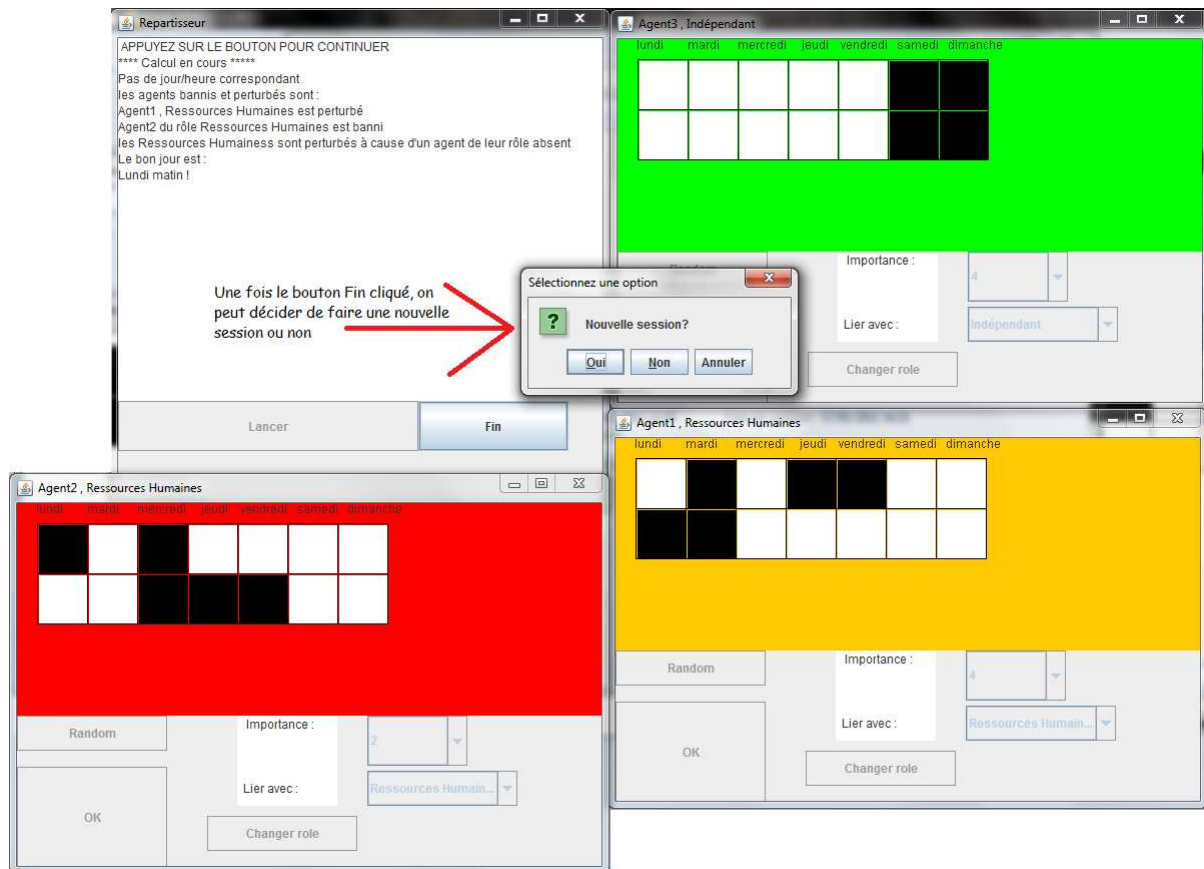
lundi	mardi	mercredi	jeudi	vendredi	samedi	dimanche

Random Importance : 4

OK Lier avec : Ressources Humain...

Changer role

Etape 10 :



Si on clique sur non, toutes les fenêtres se ferment, sinon une nouvelle fenêtre pour choisir le nombre d'agents s'ouvre après la fermeture des fenêtres précédentes.

Résumé

Ceci est un rapport d'un projet sur la programmation multi-agents. C'est une méthode de programmation adaptée à la création d'intelligences artificielles.

Le logiciel décrit ici a pour but d'aider les nouveaux utilisateurs de MadKit en leur fournissant une démonstration supplémentaire des possibilités qu'offre cette plate-forme. Ce programme simule la prise de rendez-vous d'une communauté d'individus au sein d'une entreprise, les caractéristiques des individus étant choisies par l'utilisateur.

Avec ce document, vous pouvez ainsi découvrir la conception et la création d'un logiciel multi-agents.

Summary

This is a report of a project on multi-agents programming. This is a programming method suited to the creation of artificial intelligences.

The software described here is intended to help new MadKit users by providing a further demonstration of the possibilities provided by this platform. This program simulates the making of appointments in a community of individuals within a company, where individuals' characteristics are chosen by the user.

With this document, you can discover the design and creation of a multi-agents software.