Rapport Simulation

Marion Pavaux, Hortemel Juliette

21 Mai 2021

Contents

1 Multi-agent simulation model of a complex system with the description of at least its agent, environment and interaction dimensions of the system

1.1 Les Agents

Dans notre simulation, les robots infirmiers ont pour but de récupérer les patients à l'entrée, de les amener dans les salles nécessaires à leur traitement, puis de les raccompagner à la sortie. Il existe différents types de patients, rouge, jaune et vert, correspondant à différents niveaux de gravité, et ayant besoin d'un traitement passant par un nombre de salles différent. Un patient rouge est un patient grave qui a besoin d'être traité dans 3 salles successives, un patient jaune est un patient de gravité modérée, qui a besoin d'être traité dans deux salles successives, et un patient vert est un patient non grave, ayant besoin d'être traité seulement dans une salle. Les patients arrivent au fur et à mesure du temps de manière non prédictible par les robots.

1.1.1 Les robots

Nous avons créé une classe robot, ayant pour instance la grille et l'espace dans lequel il se trouve, ainsi que son numéro d'identifiant. Un robot est équipé de nombreuses méthodes lui permettant d'interagir avec les patients, qui seront détaillées dans la partie interactions ci-dessous.

1.1.2 Les patients

La question de la représentation des patients n'a pas été évidente. Nous avons d'abord pensé à représenter les patients simplement avec une liste, qui serait mise à jour dès qu'un robot amène un patient avec lui mais cela représentait plusieurs inconvénients, à commencer par le fait qu'en simulation, il est préférable d'avoir une représentation graphique de ses agents et de pouvoir suivre leur mouvement sur la grille.
Il existe, comme mentionné plus haut, trois types de patients, les patients rouges, jaunes, et verts. Nous avons créé une classe patient avec pour instances la grille et l'espace dans lequel il se trouve et leur identifiant pour les différencier. Nous avons alors créé une classe par type de patient, cela permet de faciliter le traitement et permet d'afficher les patients différemment dans l'interface graphique (et donc de leur attribuer différentes couleurs en fonction de leur type), ces classes héritent de la classe patient.

1.1.3 Le concierge

Dans notre modèle, les patients doivent arriver petit à petit au cours de la simulation. Se pose alors la question de leur arrivée, comment ajouter des agents à la grille au fur et à mesure du temps ?

Nous avons eu l'idée de créer un agent dont le rôle est de faire venir les patients, que nous avons appelé le concierge. Le concierge est doté d'une méthode `addPatient`, qui se déclenche tous les 5 pas de temps, et qui ajoute un patient au contexte selon les probabilités suivantes :

- 20% de chance qu'un patient de gravité sévère arrive, c'est-à-dire un patient rouge.
- 30% de chance qu'un patient de gravité moyenne arrive, c'est-à-dire un patient jaune.
- 40% de chance qu'un patient de gravité légère arrive, c'est-à-dire un patient vert.
- 10% de chance qu'aucun patient n'arrive.

Le patient ajouté est alors placé par le concierge à l'entrée, au point (1,5), et est prêt à être traité par les robots. Le concierge est pour sa part positionné en (1,6) et n'a pas vocation à se déplacer lors de la simulation.

1.2 L'environnement

1.2.1 La grille

Nos robots opèrent dans un hôpital que nous modélisons par une grille de taille 12 sur 16. Cet hôpital comporte 6 salles, une entrée par laquelle arrivent les patients et une sortie par laquelle ils sortent. L'hôpital présente également des murs non franchissables par les agents, qui délimitent les salles (Figure 1).

Nous nous sommes d'abord demandé comment représenter cette grille et nous sommes intéressées à la classe `GridPoint` de Repast Simphony. La grille des zombies se construit à partir de `GridPoints`, qui ont pour attribut leur position uniquement. Notre première idée a été de construire une classe `GridPointHospital` héritant de `GridPoint`, et indiquant la salle à laquelle le point appartient. :

Il s'est cependant avéré que de changer le type de `GridPoint` dans les fonctions de Repast était difficile d'accès, et avons dû penser à une autre stratégie.
De plus, la question de l'accès aux salles se pose, où doit aller un robot quand il est dans une salle pour déposer le patient ? Nous avons choisi de définir un point dans chaque salle qui sera le point d'arrêt du robot pour simplifier le problème. Deux robots peuvent se trouver au même endroit ou se croiser sans difficultés.
Nous avons décidé de créer une classe `Salle` avec pour instances le nom de la salle, la localisation du point d'arrêt, mais également le temps de traitement, c'est-à-dire le temps nécessaire au traitement d'un patient dans cette salle. Dans un premier temps, nous n'exploitons pas cette donnée et considérons que le traitement est immédiat lorsque le patient entre dans la salle, celui peut alors immédiatement être déposé dans la salle suivante nécessaire à son traitement ou être raccompagné à l'entrée.
Les murs ne sont pas directement représentés sur la grille mais seront pris en compte par les fonctions de mouvement des robots.

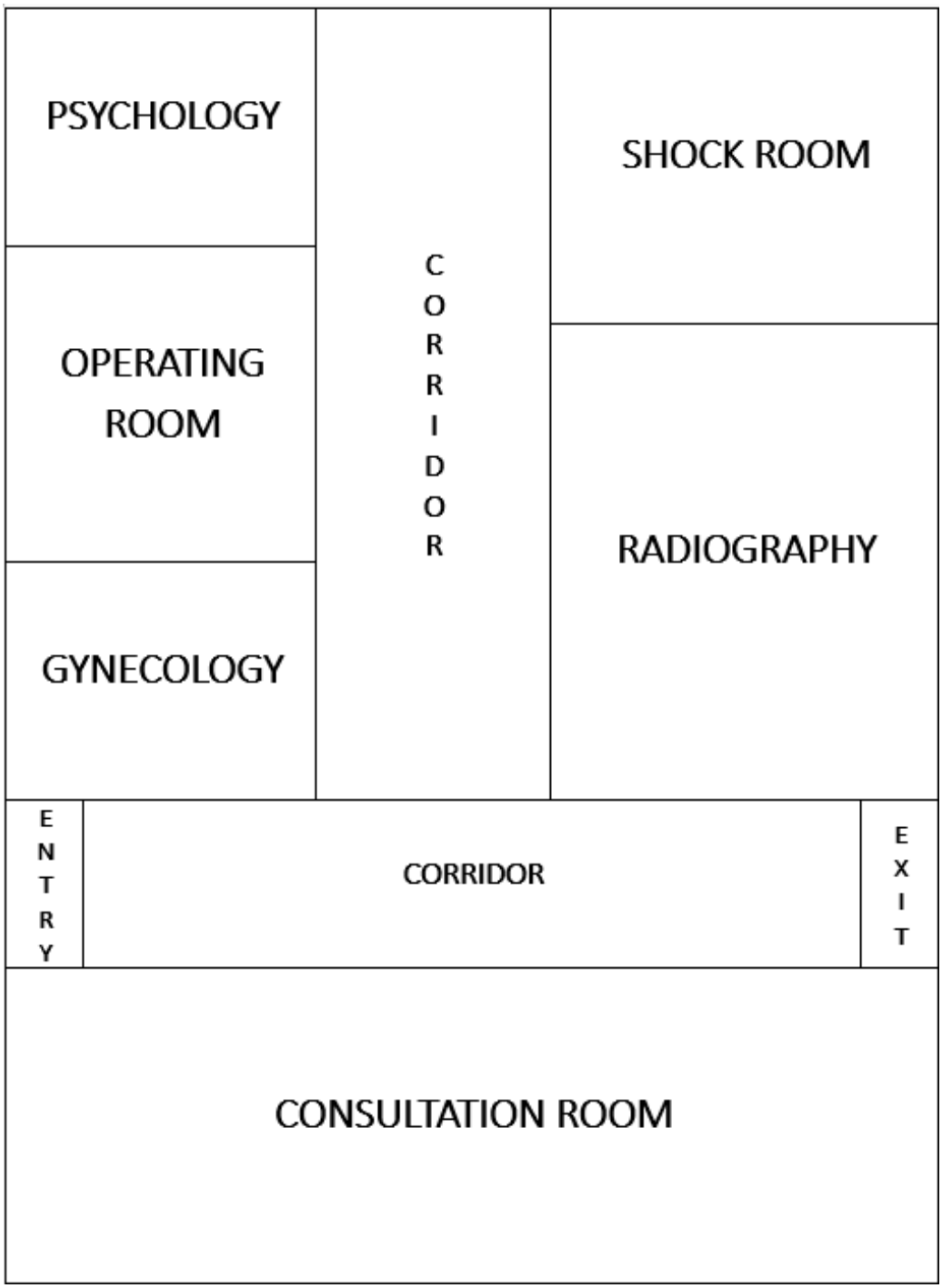


Figure 1: Plan de l'hôpital, comportant 6 salles avec une entrée et une sortie.

1.3 Les interactions

1.3.1 Les interactions robot-grille

Les robots peuvent se déplacer sur la grille pour amener les patients dans les différentes salles nécessaires à leur traitement. Cependant, ils ne peuvent pas s'y promener librement car l'hôpital présente des murs. Ces murs sont représentés avec le reste de l'agencement de l'hôpital dans le dernier rendu. Nous ne pouvons donc pas laisser nos robots parcourir les distances à vol d'oiseau dans l'hôpital. La représentation des murs étant une question délicate, nous avons décidé d'utiliser un moyen de déplacer les robots de salle en salle en leur indiquant le chemin à chaque fois. On peut en effet supposer que le robot n'a pas besoin de recalculer son chemin à chaque instant et que celui-ci "connaît déjà l'hôpital". Cependant, cela représente un grand nombre de cas à coder, nous avons donc décider de simplifier le problème en codant une méthode capable de ramener le robot en un point du couloir central (représenté par une croix rouge sur le schéma) à partir du point d'arrêt de n'importe quelle pièce, et une méthode capable d'amener le robot au point d'arrêt de n'importe quelle pièce à partir de ce même point du couloir central.
Nous avons commencé à coder ces méthodes en utilisant la fonction java `moveTo`, en utilisant un déplacement vertical et un déplacement horizontal. Cependant, la différence n'est pas visible, cela revient au même que d'utiliser un seul `moveTo`, puisque le robot effectue son déplacement en un pas de temps.

1.3.2 Les interactions robot-patient

Un robot est doté de nombreuses méthodes lui permettant d'interagir avec les patients. Il est capable de porter un patient grâce à la méthode `take` qui prend un argument de type patient et l'ajoute à la liste de patient `carry`, propre au robot. Dans notre modèle, un robot ne transporte qu'un seul patient à la fois, cependant la liste `carry` est faite de telle manière qu'un robot pourrait prendre deux patients ou plus à l'entrée sans problèmes. De manière similaire, un robot peut déposer un patient, généralement à la sortie, grâce à la méthode `drop`, qui dépose un patient, généralement à la sortie, et le retire donc de la liste `carry`. Un robot est également doté d'une méthode lui permettant de choisir un patient et de le prendre dans la foulée. Les pourcentages de chance qu'un robot prenne un type ou un autre de patient sont modulables, et différents cas peuvent être testés pour chercher la solution la plus efficace selon les critères que l'on veut maximiser (minimiser le temps moyen d'un patient dans l'hôpital, prendre en charge en priorité les patients les plus graves ...). Dans notre problème la probabilité dépend du nombre de patients de chaque type en attente dans l'hôpital. Pour appliquer ces probabilités, nous prenons un nombre au hasard entre 0 et 1 grâce à la fonction `Math.random`, et comparons sa valeur aux différentes probabilités de prendre en charge un patient de telle ou telle couleur. Pour le patient rouge, sa probabilité d'être pris ne dépend pas que du nombre de rouges mais du nombre de patients verts et jaunes. Il se peut alors que la fonction `Math.random` donne un nombre correspondant à cette probabilité alors même qu'il n'y a pas de patients rouges. Il faut bien faire attention à ce que le robot veuille toujours prendre en charge un patient qui existe en salle d'attente.
Un robot est doté d'une méthode `careScheduling`, lui permettant d'évaluer dans quelles

salles doit être traité tel patient. En réalité, et puisque nous avons trop peu de connaissances médicales, la liste des salles nécessaires au traitement s'établit de manière aléatoire, en prenant tout de même en compte le fait que la salle de consultation est plus souvent visitée que les autres, et jamais en deuxième ou troisième position. Aussi pour les patients de gravité légère et modérée, la *shock room* n'est jamais envisagée.

1.3.3 Les autres interactions

Il y a peu d'interactions robot-robot pour l'instant, si ce n'est que la liste des patients disponible est mise à jour lorsqu'un robot prend un patient. Cela pourrait représenter une piste d'amélioration du modèle.
Il n'existe aucune interaction patient-patient, ce qui est normal dans notre simulation.
Le concierge n'interagit pas avec les robots, à part par le biais de la liste des patients qui est partagée, et laisse simplement entrer les patients tous les 5 pas de temps avec les probabilités décrites plus haut.

1.3.4 Le lancement de la simulation

Pour les robots, nous avons fait le choix de regrouper les appels aux méthodes nécessaires au bon fonctionnement d'un robot dans une seule méthode `makeDecision`, que nous appelons à chaque pas de temps et qui effectue les actions suivantes selon l'état du système :

- Si il n'y a plus aucun patient dans la liste des patients en salle d'attente et que le robot ne porte personne, il renvoie un message de fin de simulation.
- Si le robot est à la sortie, il retourne à l'entrée grâce à une méthode spéciale ne passant pas par le point d'arrêt du couloir central.
- Si le robot est à l'entrée et ne porte personne, alors il choisit un patient, commence à le porter, et détermine les salles nécessaires à son traitement. Ce faisant, il l'enlève de la liste patient du concierge.
- Si le robot porte un patient et doit encore visiter des salles, alors il va à la salle suivante et la supprime de la liste des salles de parcours.
- Si le robot porte un patient mais n'a plus de salle à visiter, alors il l'amène à la sortie et le dépose. Dans ce cas le patient est aussi retiré de la simulation.

2 Improvement proposal, answer to the research question. Indicators for assessing the proposal

2.1 Améliorations possibles et implémentation

A terme, l'idéal serait d'avoir des robots capables de déposer un patient dans une salle pendant la durée de traitement, et d'aller s'occuper d'un autre patient, pour optimiser le nombre de patients pris en charge par unité de temps. La communication entre robots serait alors cruciale puisque qu'un robot x pourrait récupérer un patient amené par un autre robot y dans une autre salle, suite à un message de celui-ci donnant sa localisation et durée de traitement.
Cela paraît ambitieux et nous allons tenter d'implémenter les bases qui pourront permettre cette amélioration idéale.

2.2 Faire attendre les robots dans les salles

Nous voulons tout d'abord que nos robots soient capables d'attendre lors du traitement d'un patient dans la salle concernée, sans les déposer, mais en respectant bien la durée de traitement. Nous commençons ainsi par introduire la notion de durée de traitement (en réalité introduite plus tôt mais non exploitée), chaque salle possède une durée de traitement bien précise accessible grâce à la méthode `getDureeTraitement`. Nous introduisons également pour chaque robot, les instances d'entier `clock` et d'entier `TreatmentPeriod`, qui vont permettre de respecter la durée d'attente des salles.
Nous ajoutons donc une contrainte supplémentaire au robot pour pouvoir amener le patient qu'il porte à la salle suivante : il faut que son horloge soit supérieure à la durée de traitement. Lorsque son horloge est bien égale à la durée de traitement, elle est alors réinitialisée à 0 et celui-ci peut continuer son chemin. L'horloge est incrémentée lorsque le robot ne peut faire aucune des autres actions, cela signifie qu'il doit attendre dans la salle dans laquelle il est. Le robot n'a pas besoin d'attendre lorsqu'il est à l'entrée, on fixe alors son horloge à 100 pour être sûr que celui-ci n'attendra pas avant d'aller dans une autre salle, où son horloge se réinitialisera en entrant.

2.3 Faire communiquer les robots

Nous voulons maintenant que nos robots puissent communiquer entre eux. Nous allons faire en sorte que, lorsqu'un robot prend en charge un patient, celui-ci envoie les informations du patient et sa localisation à tous les autres robots. La localisation n'est pas particulièrement intéressante pour notre modèle actuel mais deviendra cruciale si les robots peuvent "s'échanger" des patients et les laisser seuls durant leur durée de traitement.
Nous commençons par créer une classe de message `RobotMessage`, qui contient les données d'un patient. Nous ajoutons une donnée `mailbox` aux robots. Nous ajoutons ensuite de nouvelles méthodes à nos robots. Nous ajoutons la méthode `hello`, qui permet d'envoyer

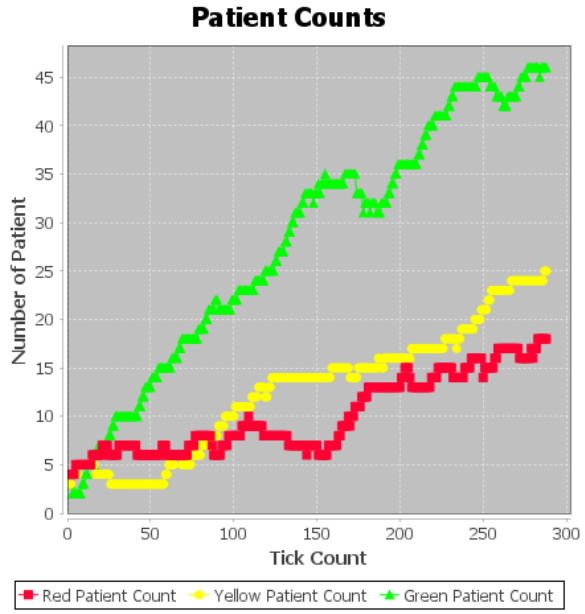
un `RobotMessage` aux autres robots, ainsi que la localisation du robot. Cette méthode utilise une autre méthode `send`, qui envoie le message, et qui déclenche la méthode `receive` chez les autres robots. La méthode `receive` permet d'ajouter un message dans la `mailbox` d'un robot. Il nous suffit alors de déclencher la méthode `hello` directement après avoir effectué l'accueil et la détermination des salles nécessaires au traitement d'un patient.

2.4 Indicateurs les propositions d'amélioration

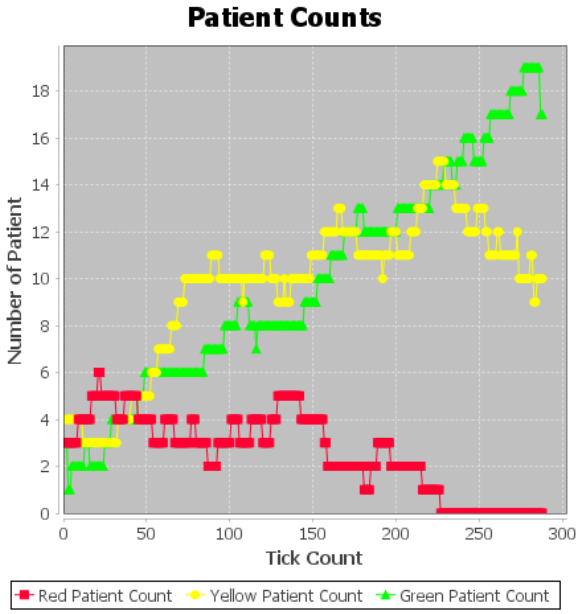
La figure 2 nous donne les relations entre le nombre de patients et la fréquence d'arrivée de ceux-ci. Nous avons donc évalué ces relations pour une arrivée toutes les 10min (2 ticks) (Figure 2a), 20min (Figure 2b), 30min (Figure 2c) et 40min (Figure 2d), avec deux robots dans une journée de travail (288 ticks). Pour 10 minutes (Figure 2a), les patients ne s'en sortent pas, les compteurs explosent. A partir de 20 minutes (Figure 2b), les patients graves sont traités au fur et à mesure de la journée, bien que pas très rapidement, et les patients à gravité modérée ne sont pas assez traités. A partir de 30 minutes, la situation redevient stable pour tous les patients (Figure 2c and 2d).

Nous avons donc vu qu'en cas de forte affluence, les robots n'arrivent pas à gérer les patients. Dans des urgences à plus forte influence, on peut rajouter des robots et voir s'ils arrivent à gérer les patients. La figure 3 montre l'influence du nombre de robots sur le nombre de patients dans l'hôpital au cours du temps, avec respectivement 6 (Figure 3a), 8 (Figure 3b), 10 (Figure 3c), et 12 (Figure 3d) robots pour une arrivée de patients toutes les 5 minutes. On voit que pour 6 (Figure 3a) et 8 robots (Figure 3b), les patients graves sont globalement bien traités, avec pour 8 robots, des patients modérés également traités. Cependant, le nombre de patients verts explose. A partir de 10 robots (Figure 3c), du fait des probabilités, les patients légers sont bien trop privilégiés. Une amélioration possible serait donc de tenir compte dans les probabilités de sélection des patients, du nombre de robots.

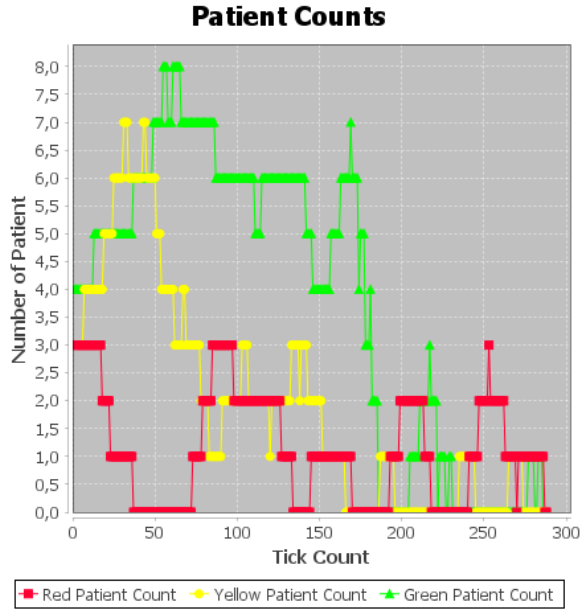
Un robot a besoin de seulement un tick (5 minutes) pour se déplacer vers une autre salle. Au lieu de cela, il attend, dans le meilleur des cas 15 minutes (salle de consultation), et dans le pire des cas une heure (salle d'opération) avec le patient dans sa salle. Il est toujours plus rentable pour le robot de se déplacer à l'entrée (5 min), de choisir un patient et de le porter (5 min), et de l'emmener dans sa première salle (5 min) plutôt que d'attendre avec son patient dans la salle. Il peut également être plus rentable d'attendre la fin de traitement d'un autre patient, d'aller le chercher (5 min) et de l'emmener à sa prochaine salle ou alors à la sortie (5min). Les salles d'opérations sont de vrais gouffres à efficacité, et nous pensons que si nous avions pu mettre en place l'amélioration jusqu'au bout, nous aurions pu garder deux robots pour une fréquence de seulement 10 minutes d'intervalle entre chaque patient.
Nous avons fait la première étape de l'amélioration qui était l'attente dans les salles et la communication entre robots. La deuxième étape aurait été de lâcher les patients à l'arrivée dans une salle, et de choisir entre prendre un patient à l'entrée et aller rechercher un de ses patients qui a fini ou presque fini son traitement. La troisième étape aurait été de pouvoir également aller chercher des patients que d'autres robots avaient déposé cette fois-ci, qui ont également fini ou presque fini leurs traitements.



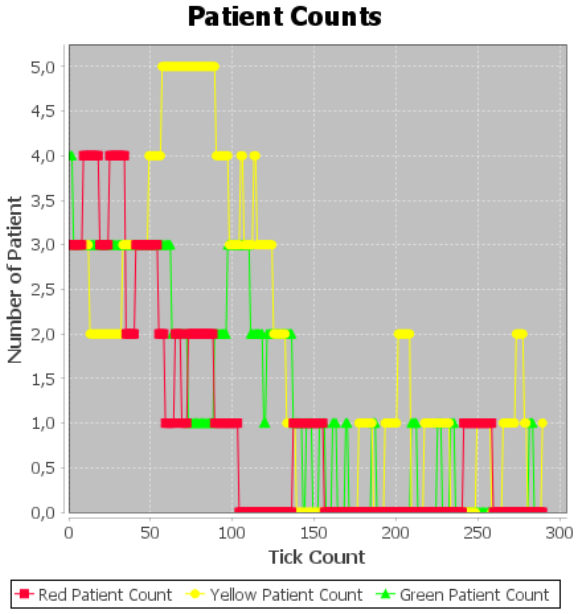
(a) 10 minutes



(b) 20 minutes

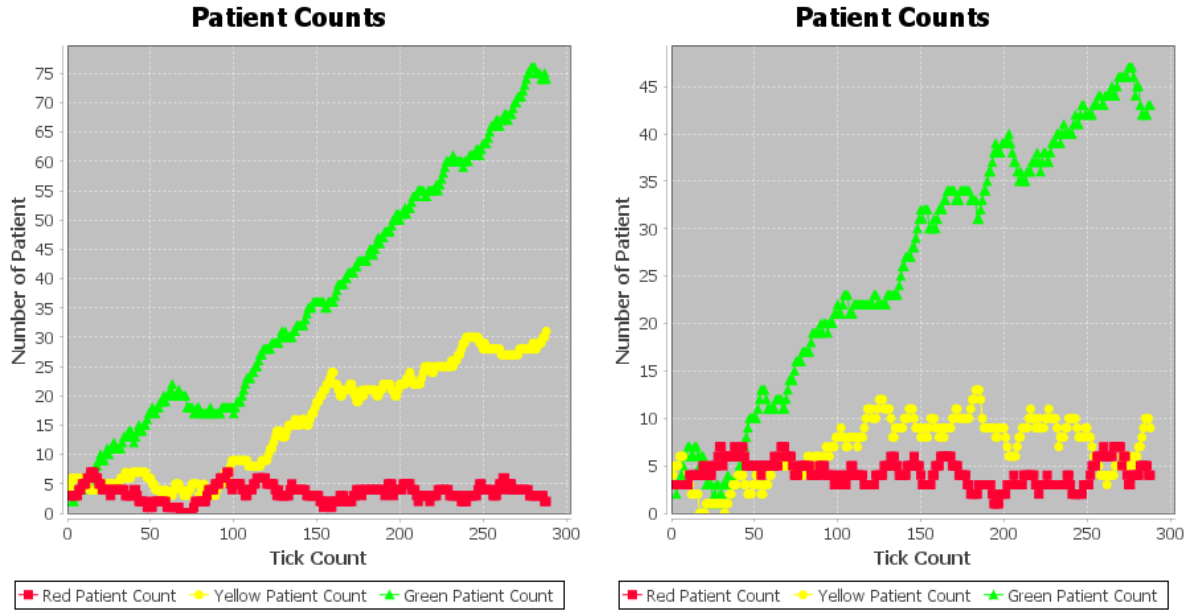


(c) 30 minutes



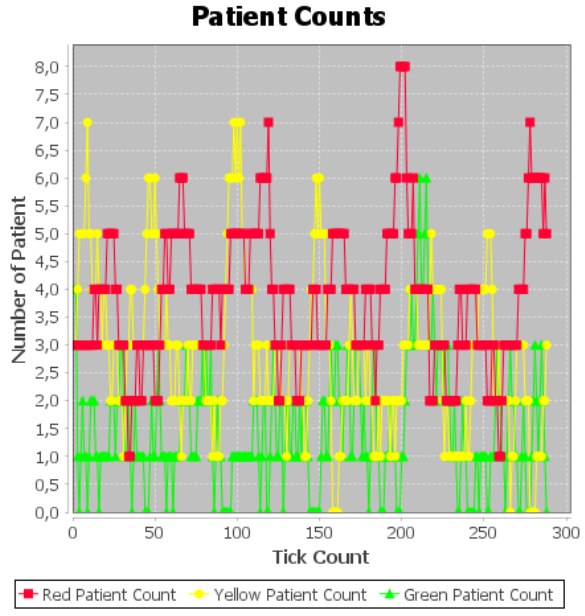
(d) 40 minutes

Figure 2: Influence du temps d'arrivée entre deux patients sur l'évolution du nombre de patients dans l'hôpital en fonction du temps. Pour 2 robots, (a) 10 minutes (b) 20 minutes (c) 30 minutes et (d) 40 minutes.

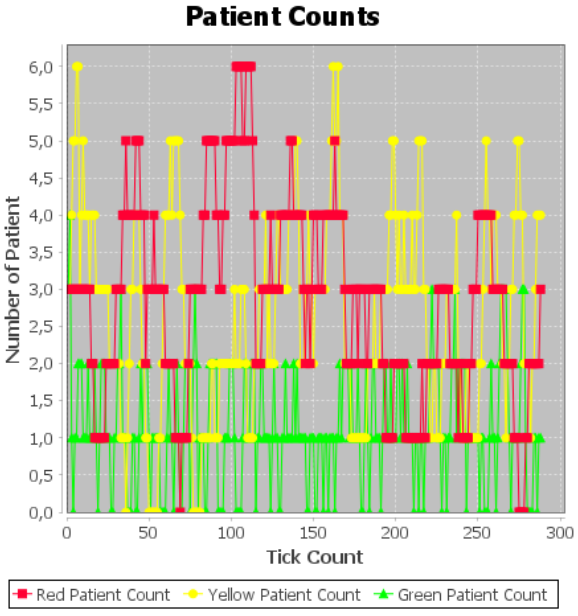


(a) 6 robots

(b) 8 robots



(c) 10 robots



(d) 12 robots

Figure 3: Influence du nombre de robots actifs sur l'évolution du nombre de patients dans l'hôpital en fonction du temps. Pour 5 minutes d'intervalle entre l'arrivée de deux patients, (a) 6 robots (b) 8 robots (c) 10 robots et (d) 12 robots.