Tarek DALATI
Nathan WINGER

Musée, Voleur et Gardes

Projet de Vérification de Systèmes

2018-2019

Introduction

Le système que nous avons décidé de modéliser représente un voleur dans un musée. Le but de ce voleur est de sortir du musée tout en évitant les différents gardes mobiles et en optimisant son parcours afin de récupérer les objets ayant le plus de valeur.

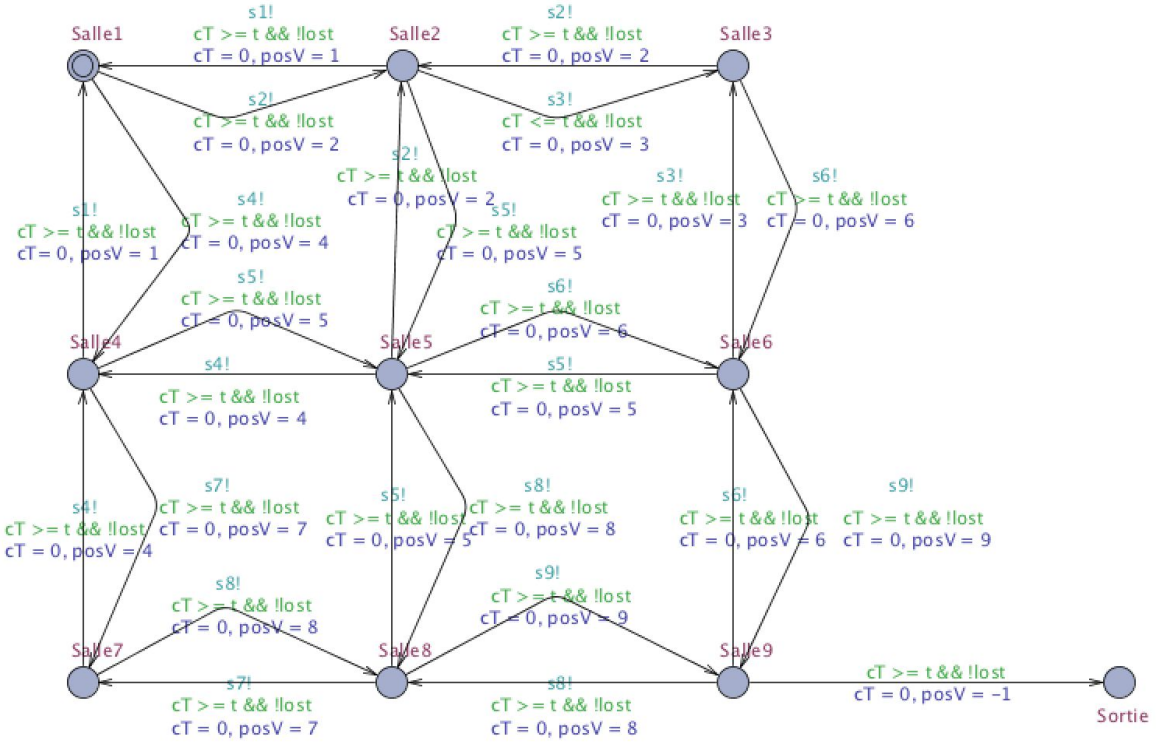
Nous avons mis en place plusieurs règles à respecter :

- dès le moment qu'un garde se trouve dans la même pièce que le voleur, le voleur est attrapé
- les gardes se déplacent de façon déterministe
- le voleur met 30 secondes pour passer d'une salle à une autre
- les objets possèdent un poids, le poids affecte la vitesse de déplacement du voleur
- il existe un objet bonus permettant de réduire la vitesse de passage d'une salle à une autre de 20 secondes

Ainsi, nous nous sommes mis dans une optique de game design afin de réussir, grâce à la modélisation et sa vérification, d'envisager un jeu et d'ajuster sa difficulté.

I. Modélisation du musée et du parcours possible du voleur

Le premier objectif a été de modéliser un musée représentant les choix de parcours du Voleur, nous permettant dans un premier temps de vérifier si le voleur avait la capacité de sortir du musée dans le temps impartis.



*Système du **Voleur** (représentant les états possibles du voleur dans le musée)*

Description des variables :

- cT : timer du voleur permettant de modéliser le temps écoulé dans l'état actuel
- lost : booléen permettant de savoir si la partie est perdue ou non (si le voleur est attrapé ou non)
- t : constante représentant le temps de passage d'une salle à une autre (initialisé à 30 secondes)
- posV : entier représentant le numéro de la salle où se trouve le voleur (-1 si le voleur est sorti)
- gT : temps global écoulé depuis le début de la simulation
- tempsLimite : temps impartis au voleur afin de sortir du musée, s'il le dépasse, la partie est perdue.

Description des événements :

- s1 à s9 : événements générés lors du passage d'une salle à une autre

Nous avons décidé de représenter chaque salle du musée ainsi que la sortie par un état. Le voleur se déplace d'une salle à une autre dans un laps de temps minimum à t secondes. Il continue de se déplacer tant qu'il n'est pas attrapé.

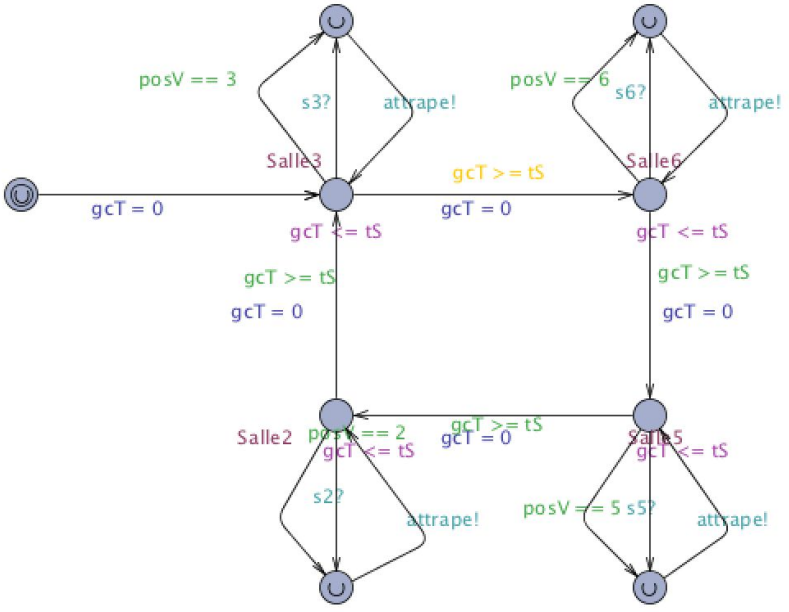
Vérification :

Pour vérifier que le voleur puisse sortir du labyrinthe, nous avons utilisé la formule suivante :

- E<> (Voleur.Sortie && gT <= tempsLimite)

Si le temps de passage d'une salle à une autre est de 30 secondes et que le temps impartis est supérieur ou égal à 120 secondes, le voleur peut sortir du musée.

II. Modélisation d'un premier garde



Système du Garde 1 (représentant la ronde prédéfinie du garde)

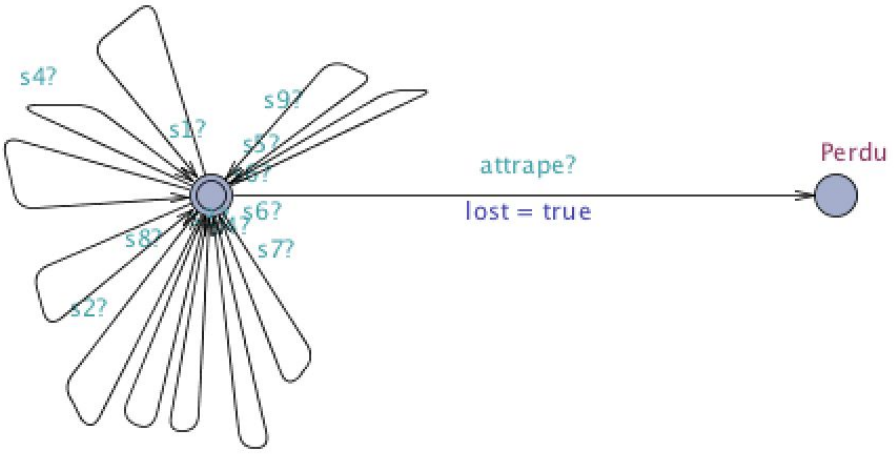
Description des variables :

- gcT : timer du garde permettant de modéliser le temps écoulé dans l'état actuel
- tS : temps minimum de surveillance dans une salle

Description des événements :

- attrape : événement déclenché quand le garde voit le voleur

Ainsi, pour que le voleur soit bel et bien attrapé lors du déclenchement de l'événement et pour simplifier le système du voleur, nous avons ajouté un automate observateur permettant le changement de statut du voleur.



Système de l'automate observateur, arbitre du jeu

Cet automate permet de vérifier si la partie est perdue ou non. Les événements receveurs sont là pour rendre possible les transitions du voleur. (sinon les transitions des événements émetteurs ne peuvent pas être prises)

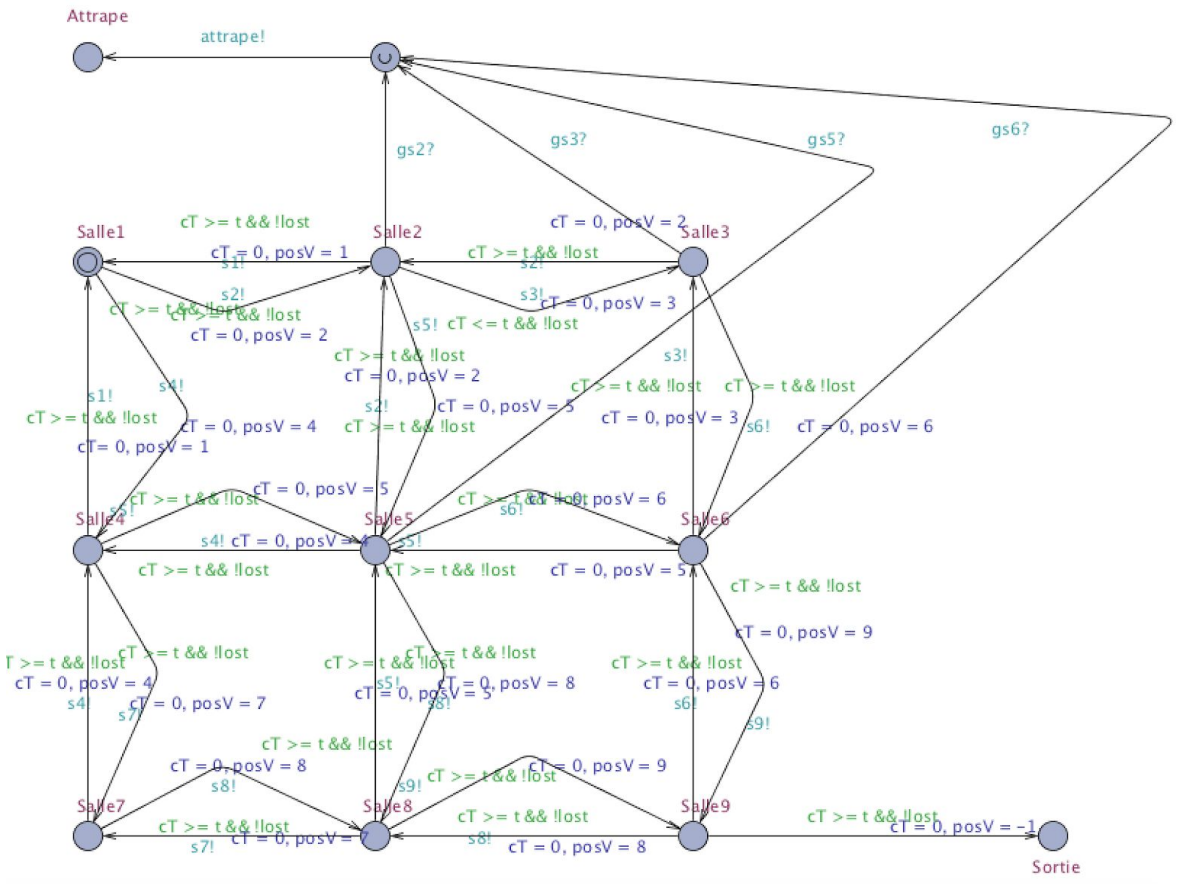
Vérification :

- E<> (Voleur.Sortie && !Observateur.Perdu && gT <= tempsLimite) -- OK
  cette vérification nous permet de savoir s'il est possible pour le voleur de sortir du labyrinthe, sans se faire attraper et dans le temps impartis.
- E<> (Voleur.Salle3 && Guard1.Salle3) && !Observateur.Perdu -- OK
  cette vérification nous permet de savoir s'il est possible que le voleur se trouve dans la même salle que le garde et que la partie ne soit pas perdue

Ici nous avons rencontré un problème : si le garde et le voleur se trouvent dans la même salle, nous voulions que le voleur soit directement attrapé et la partie perdue. Nous avons alors détecté l'origine du problème :

> *Quand un voleur rentre dans la pièce où se trouve le garde, le voleur est bien attrapé mais quand il s'agit du garde qui rentre dans la pièce du voleur, le voleur n'est pas forcément attrapé.*

Nous avons dû revoir notre système pour corriger ce problème. Nous avons utilisé le même principe d'événements que pour le voleur : dès que le garde entre dans une nouvelle pièce, un événement est généré (gs3, gs6, gs5, gs2) et automatiquement le voleur entre dans un état attrapé.



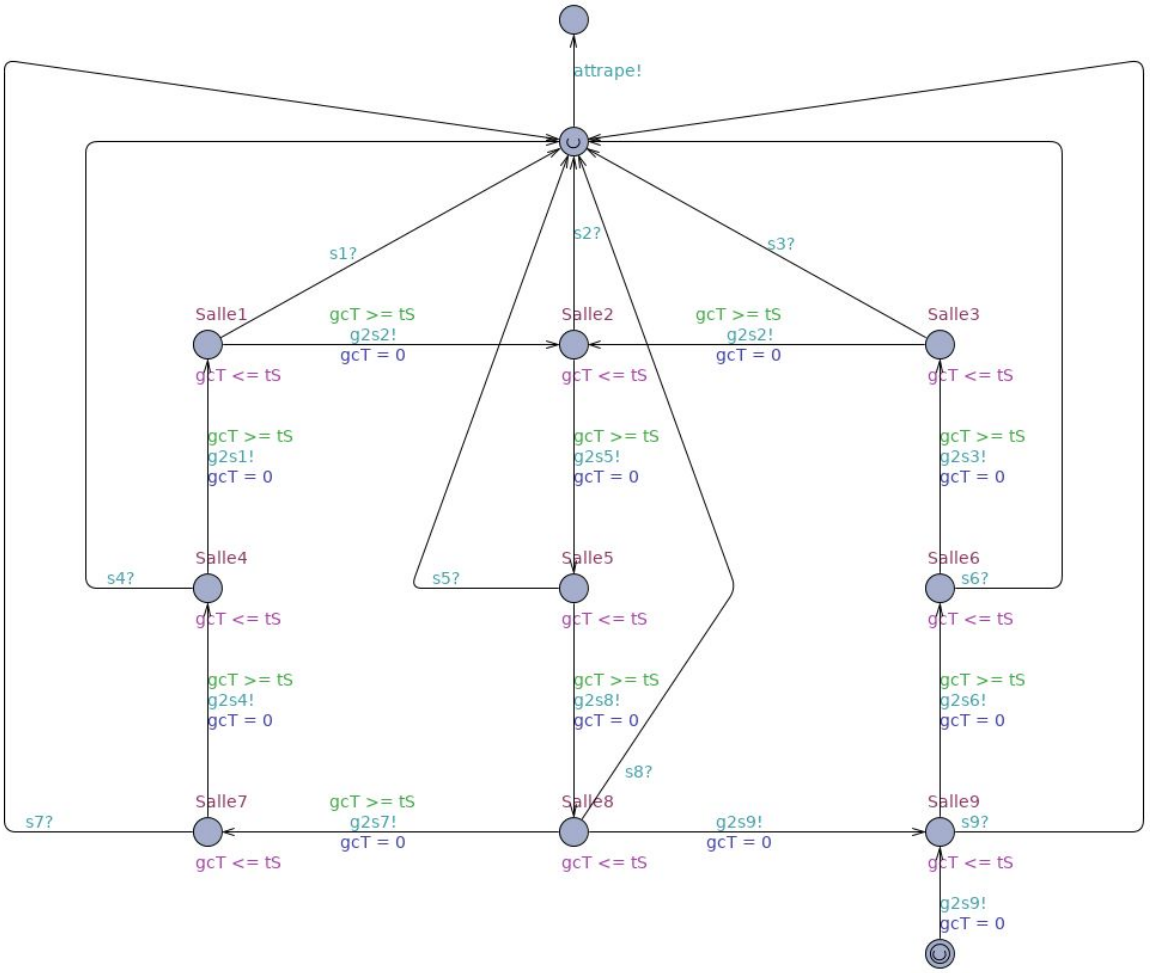
Système du voleur corrigé

Dorénavant, la vérification suivante est validée

- E<> (Voleur.Salle3 && Guard1.Salle3) && !Observateur.Perdu -- KO

III. Modélisation d'un second garde

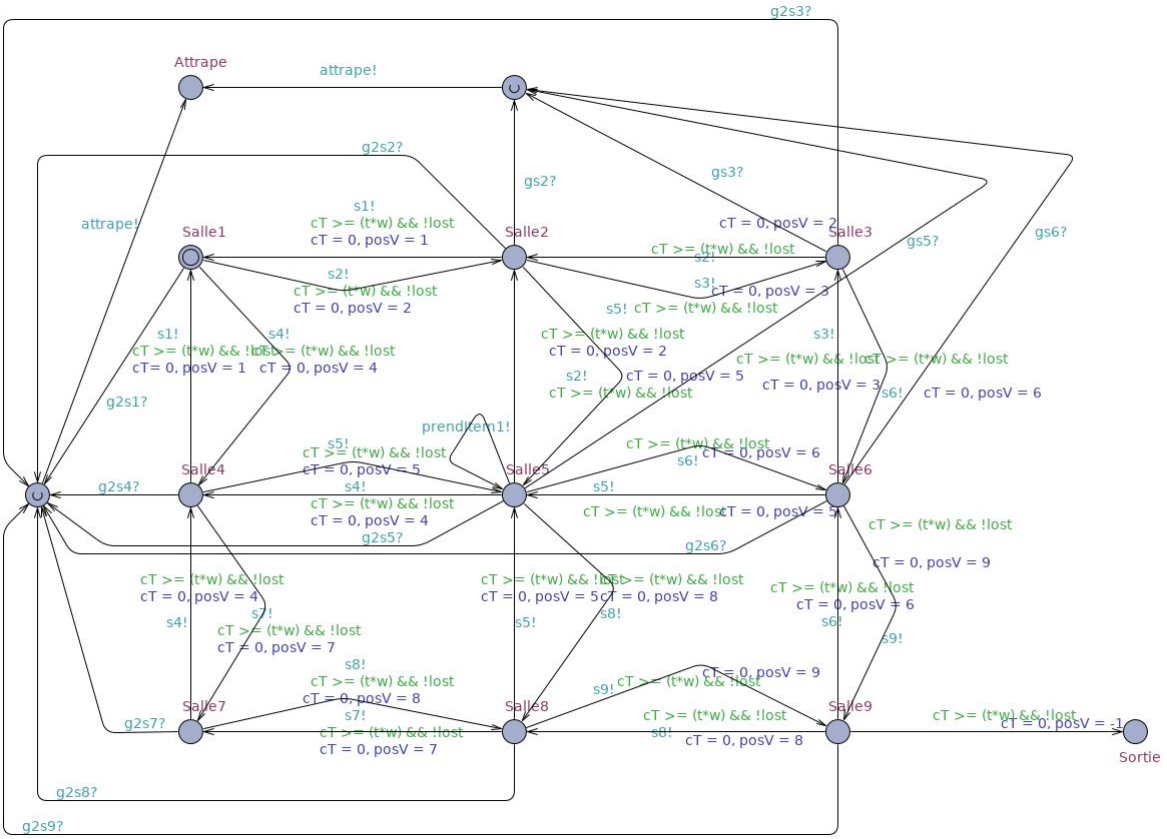
Une fois ce premier garde bien modélisé, nous avons décidé de mettre en place un second garde qui parcourt les différentes salles du musée.



Système du garde 2

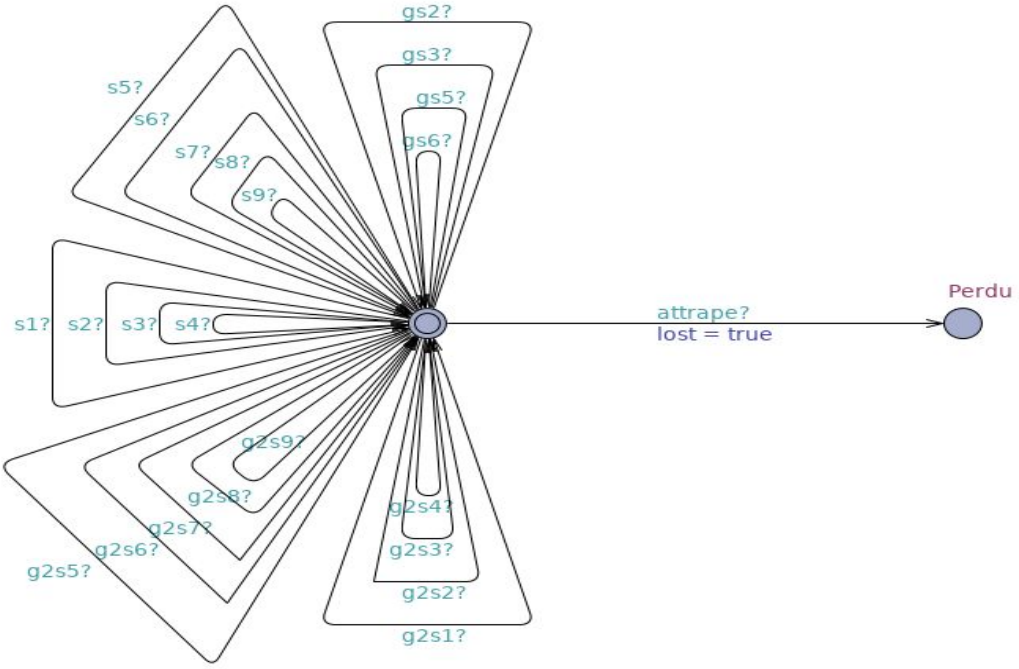
De même que pour le premier garde, nous avons utilisé les variables gcT et tS pour le timer de l'état du garde ainsi que le temps de surveillance d'une salle du musée.
De plus, nous avons aussi utilisé des événements qui sont déclenchés lorsque le deuxième garde entre dans les différentes salles: g2s1 est déclenché si le garde entre dans la salle 1, g2s2 pour la salle 2, etc.

Voici l'automate du voleur obtenu après l'ajout du deuxième garde:



Système du voleur amélioré

Pour finir, nous avons mis à jour l'observateur en ajoutant les événements du second garde:



Automate observateur pour le statut de la partie

Nous avons testé la validité de la vérification suivante:

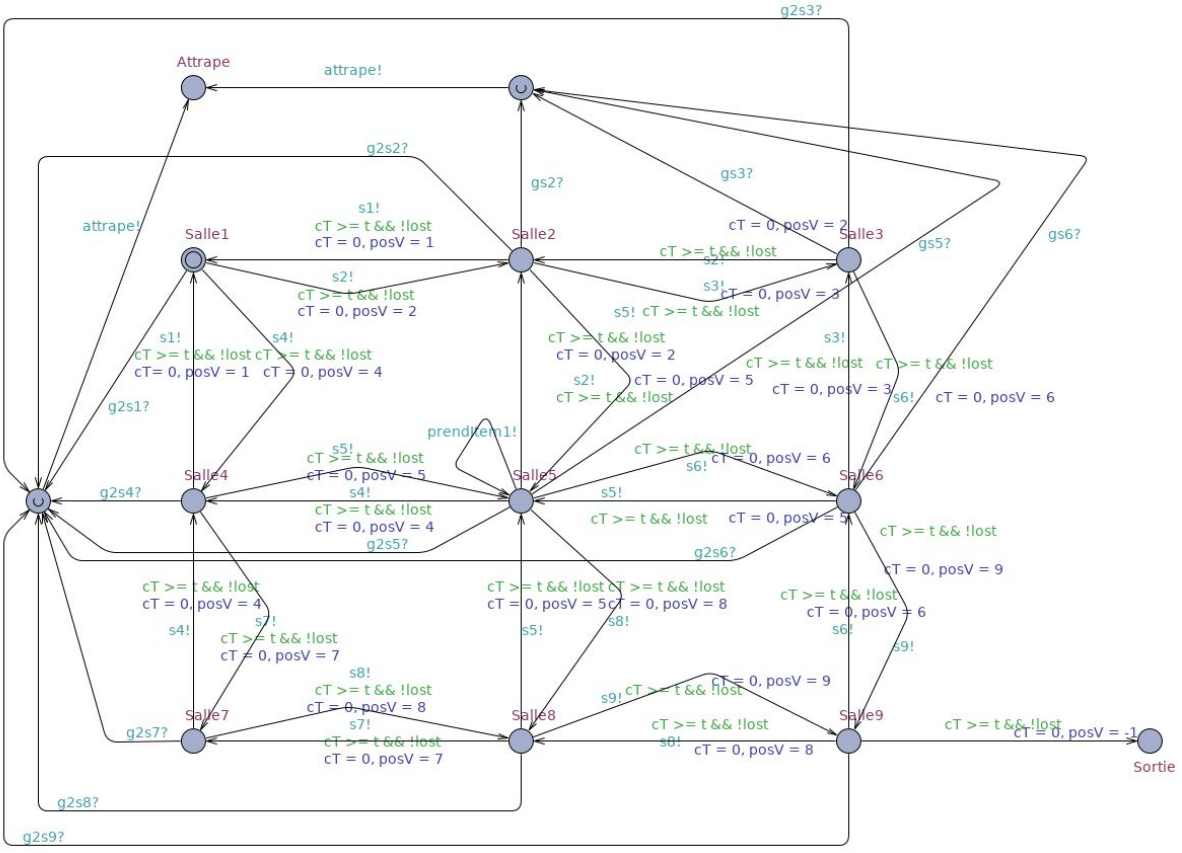
E<> (Voleur.Salle3 && Guard2.Salle3) && !Observateur.Perdu -- KO

Celle-ci nous donne bien le bon résultat.

IV. Ajout des objets

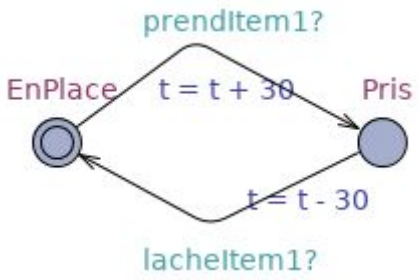
Jusqu'à présent, nous avons modélisé le voleur ainsi que les 2 gardes. Puis nous avons décidé de mettre en place des objets dans les salles du musée. Chaque objet possède un poids. Lorsque le voleur décide de prendre un objet, sa vitesse se réduit. En effet, la vitesse du voleur diminue en fonction du poids des objets qu'il porte: pour chaque objet porté, le voleur met 30 seconde de plus pour sortir d'une salle.

On obtient donc l'automate suivant:



Système du voleur après ajout d'un objet dans la salle 5

Par exemple, l'événement prendItem1! sera déclenché si le voleur prend l'objet qui se trouve dans la salle 5. De plus, nous avons fait un automate indiquant si un objet a été pris ou non:



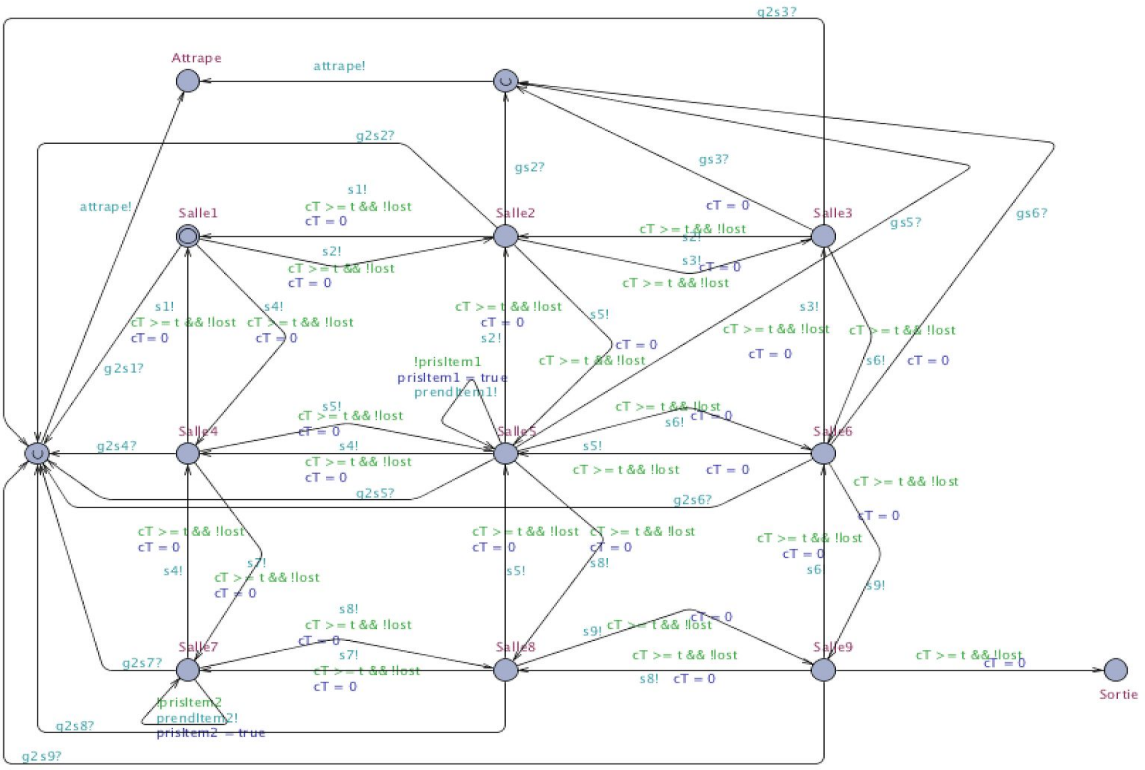
Nous remarquons bien que le temps pour sortir d'une salle augmente de 30 secondes à chaque fois qu'il prend un objet.

La vérification suivante nous a permis de tester cette formule:
E<> (Voleur.Sortie && !Observateur.Perdu && Item1.Pris && gT <= tempsLimite) -- OK

Avant d'arriver à cette version fonctionnelle, nous avons rencontré quelques problèmes. Lorsque nous voulions réduire la vitesse du voleur, nous la mettions à jour avec la condition suivante disposée sur chacun des arcs de passage d'une salle à une autre :

$cT >= t * w$

avec t étant le temps pour sortir d'une salle, w le poids de l'objet et cT le temps courant passé dans une salle. Or nous avions des problèmes dû à cette condition, il nous était impossible de vérifier, la vérification prenant un temps inconsidérable. Pour remédier à ce problème, nous avons modifié la structure de notre système modifiant la vitesse du voleur : nous avons laissé la condition $cT >= t$ et incrémenté t de 30 secondes dans l'automate de l'objet directement et nous avons enlevé la variable de poids afin d'alléger le système.

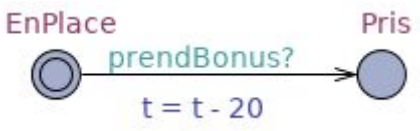


Système du Voleur nettoyé avec le deuxième objet en salle 7

Afin d'ajouter un peu de piment, nous avons ajouté un deuxième objet en salle 7. La vérification de la possibilité du joueur de gagner dans le temps impartis avec les deux objets pris ici ne passait pas. Le temps de passage d'une salle à une autre était trop élevé et les gardes avaient le temps de se déplacer sur plusieurs salles avant que le voleur ne puisse se déplacer que d'une seule. Nous avons donc revu la mise à jour du temps de passage d'une salle à une autre à la baisse (incrémentation de 15 secondes pour chaque objets seulement) afin de rendre le jeu possible.

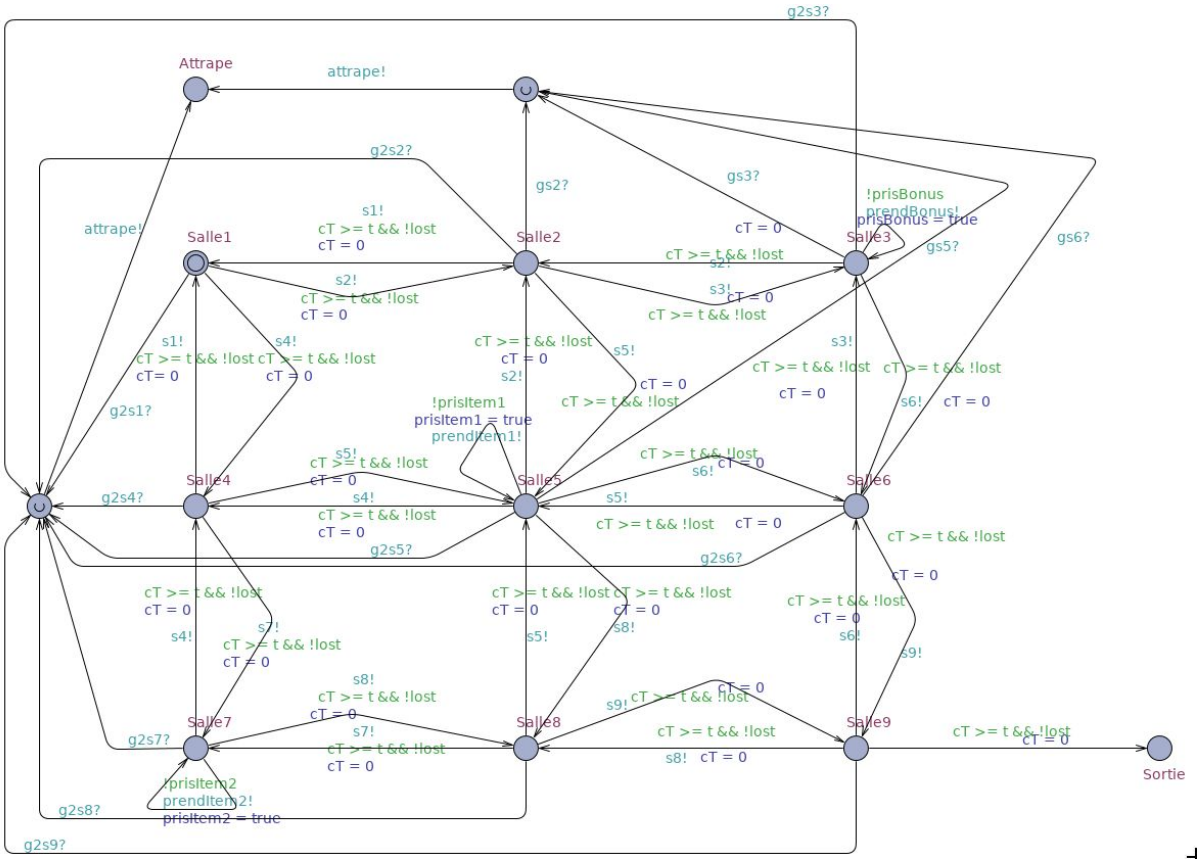
V. Ajout d'un bonus

Etant donné le problème souligné précédemment, nous avons décidé d'implémenter une autre alternative que de réduire l'augmentation du temps de passage. Nous avons ajouté un objet bonus permettant au voleur de réduire de 20 secondes le temps de déplacement d'une salle à l'autre du musée.



Système de l'objet bonus

L'utilisation de l'événement *prendBonus* permet de détecter si le voleur a pris l'objet bonus ou non. L'objet est ajouté au système du voleur à la salle 3.



Système du voleur avec l'ajout du bonus dans la salle 3

Comme pour les objets, nous avons utilisé un booléen *prisBonus* indiquant si le bonus a déjà été pris par le voleur. Cela évite qu'il prenne ce bonus plusieurs fois.

Une fois la mise en place de l'objet bonus effectuée, il n'est plus nécessaire de réduire le temps de déplacement d'une salle à l'autre du voleur. Le bonus permet au voleur de prendre les deux objets et de sortir du musée sans se faire attraper par les gardes.

La vérification suivante est maintenant fonctionnelle :
E<> (Voleur.Sortie && !Observateur.Perdu && gT <= tempsLimite && Item1.Pris && Item2.Pris) -- OK

VI. Vérifications

Voici la liste des vérifications réalisées au cours de ce projet :

- **E<> (Voleur.Sortie)** : vérification que le Voleur puisse atteindre la sortie
- **E<> (Voleur.Sortie && gT <= tempsLimite)** : vérification que le Voleur puisse atteindre la sortie dans le temps impartis
- **E<> (!Observateur.Perdu)** : vérification que le Voleur puisse ne pas se faire attraper par les gardes
- **E<> (Voleur.Salle3 && Guard2.Salle3) imply Observateur.Perdu** : vérification que le Voleur ne puisse pas se trouver dans la salle 3 en même temps que le garde 1 sans se faire attrapper (généralisé à toutes les salles et tous les gardes)
- **E<> (Voleur.Sortie && !Observateur.Perdu)** : vérification que le Voleur puisse sortir du musée sans se faire attrapper
- **E<> (Voleur.Sortie && !Observateur.Perdu && Item2.Pris && Item1.Pris)** : vérification que le Voleur puisse sortir du musée, sans se faire attrapper et avec tous les objets
- **E<> (Voleur.Sortie && !Observateur.Perdu && gT <= tempsLimite && Item1.Pris && Item2.Pris)** : vérification que le Voleur puisse sortir du musée, sans se faire attrapper, avec tous les objets et dans le temps impartis

Conclusion

La modélisation et la vérification de notre musée, du voleur, des gardes et des objets nous a permis la modélisation du fonctionnement d'un jeu de sorte que le joueur, incarnant le voleur, puisse gagner selon des règles définies. Cette modélisation a été faite avec le logiciel Uppaal. Malgré le fait que nous n'ayons pas trouvé ce logiciel convénient pour le travail en équipe car beaucoup de conflits sont à corriger lors de l'utilisation du gestionnaire de versions Git, Uppaal a été agréable à utiliser pour sa simplicité et ses performances.

Ce projet nous a permis de constater la nécessité d'une étape de modélisation lors de la réalisation de logiciels ou de jeux. Nous avons découvert que l'utilisation de la vérification pouvait permettre, au delà de la recherche de bugs et d'erreurs critiques, à équilibrer un jeu et améliorer sa jouabilité.