Compte-rendu projet FSM

Alves Dos Reis Emmanuel

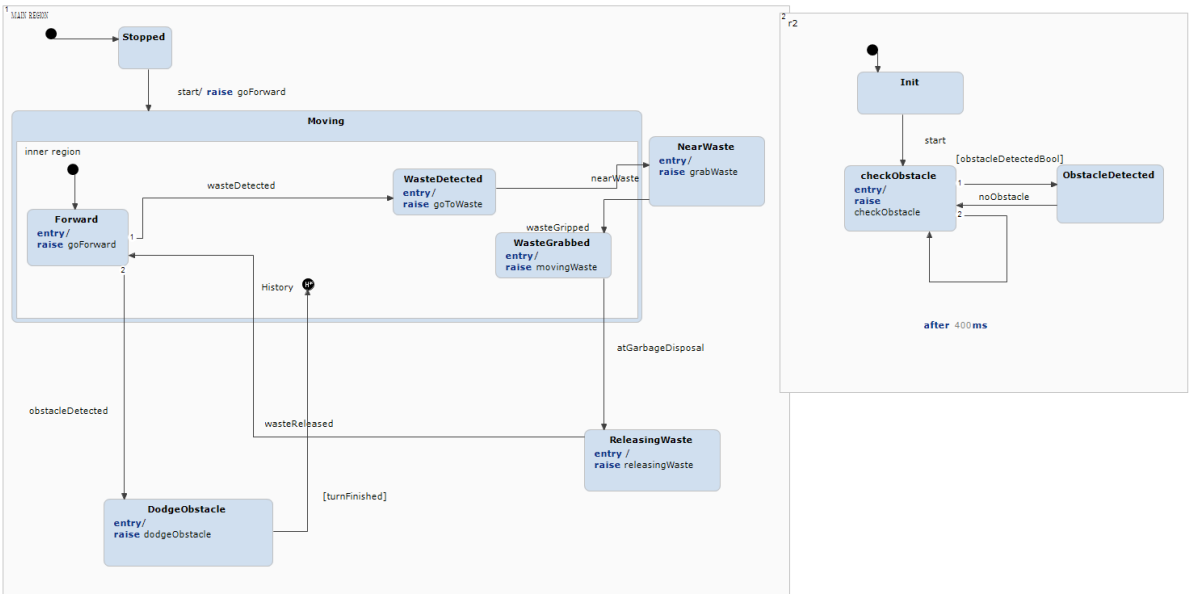
Ferdaouss Chakroun

I/Le travail réalisé

En premier lieu, nous avons essayé de réaliser un MVP du produit demandé mais nous avons été confrontés à de sérieux problèmes. Par exemple, notre robot s'arrêtait lorsque nous essayions de le faire pivoter. Ou encore lorsque nous lui demandions de reculer puis d'avancer, il reculait bien mais il n'avançait plus, même lorsque le programme montrait clairement que la fonction goForward a été appelée. Pour contourner le problème nous avons essayé de supprimer l'idée de revenir légèrement en arrière avant de tourner. Mais un deuxième problème est apparu. Lorsque le robot essayait de tourner la simulation s'arrêtait et plus rien ne répondait. Nous voulions au début implémenté le MVP ainsi que la prise en charge des déchets cependant n'arrivant même pas à produire le MVP, même vers la fin des séances encadrées nous avons vite abandonné l'idée de réaliser des options. Donc même si la statechart indique comment nous comptons traiter la partie pour gérer les déchets il n'y a en réalité pas de code associé. Pour nous assurer d' au moins livrer un MVP nous avons décidé d'utiliser un autre statechart qui ne présente pas ces problèmes. Finalement nous avons complètement changé de statechart et allons en conséquence vous présenter notre travail dans son intégralité (nos deux statechart ainsi que les tests succinct de V&V).

II/ Les statechart

Notre première version



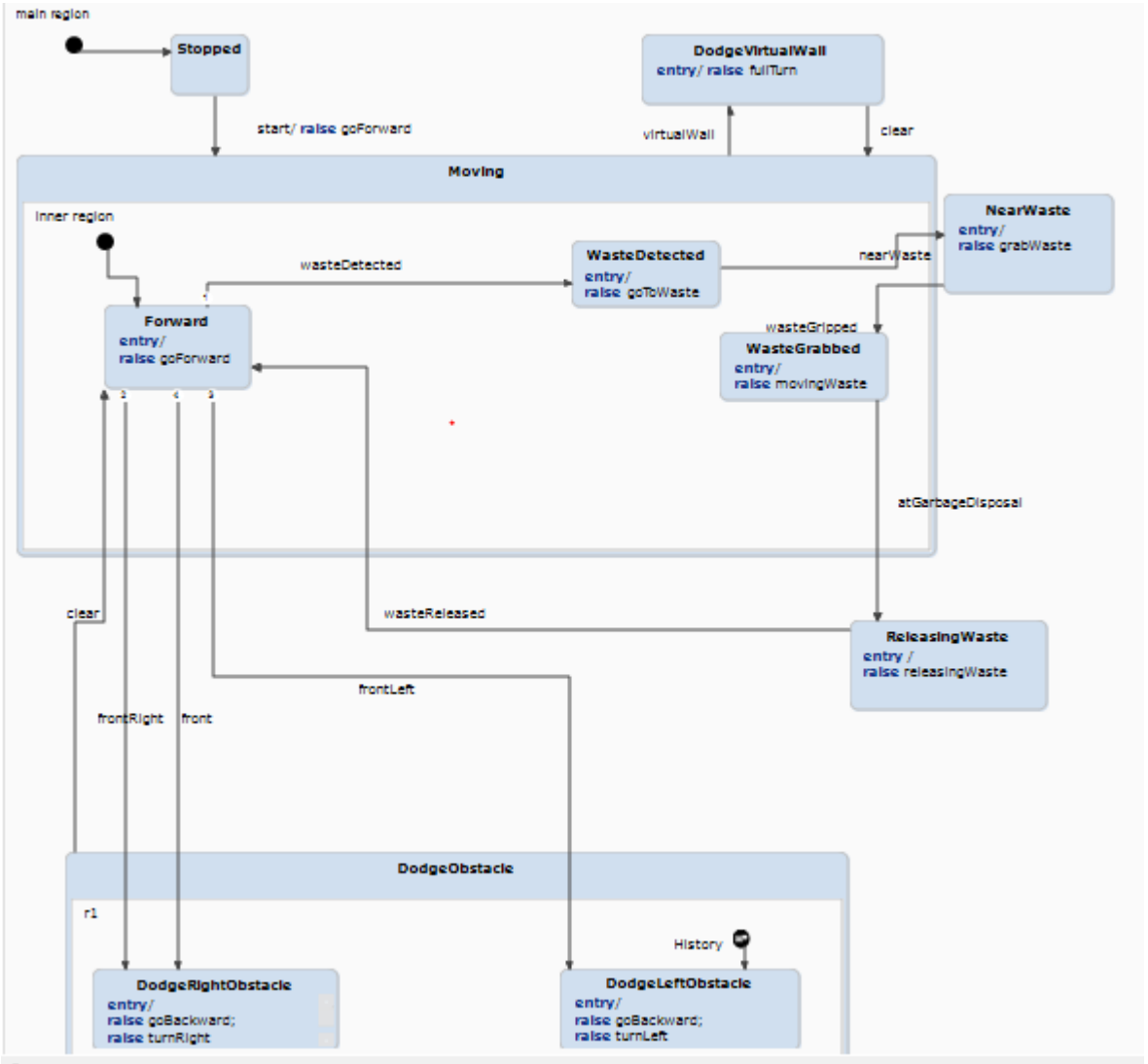
Voici le principe de construction de notre Statechart. Nous avons pensé à créer une section principale (ici Main Région) qui va réagir en fonction d'événements envoyés depuis d'autres régions en parallèle (ici r2). Ainsi pour la détection d'obstacle par exemple, nous demandons en parallèle de regarder toutes les 400 ms s'il y a un obstacle (checkObstacle), si c'est le cas alors on envoie l'évènement ObstacleDetected qui va déclencher une procédure d'évitement et on arrête de regarder les obstacles, lorsque cette procédure d'évitement est terminée, on revient à l'état précédent dans les deux régions. On recommence à regarder s'il y a des obstacles tous les 400 ms et on revient sur Forward. On va donc vérifier en parallèle en permanence s'il y a un obstacle et on réagit si c'est le cas, ce principe peut s'étendre à l'option que nous voulions implémenter qui est de gérer les déchets. Cette construction a plusieurs avantages, elle est facile à comprendre conceptuellement et permet de mieux organiser le contrôle. Elle permet également de s'étendre sur les options assez facilement. Cependant, elle pose un gros problème de synchronisation, et de communication entre le code et la statechart (qui doit envoyer l'évènement et quand), qui est sûrement la cause première de notre échec quant au développement du MVP.

Une autre solution envisageable était de demander dans le code de regarder s'il y a un obstacle, c'est-à-dire dans notre état Forward de lui demander de regarder s'il y a un obstacle ceci supprimerait les états parallèles. L'avantage est qu'il n'y a du coup aucune synchronisation nécessaire. Mais le plus gros désavantage est qu'une partie du contrôle se retrouvera dans le code et non pas dans la statechart, et également que l'ajout d'extension éventuelle risque de complexifier grandement le contrôle affixé au code.

En comparant ces deux solutions nous avons opté pour la première dans un premier temps, en pensant qu'il valait mieux utiliser la structuration de la statechart ainsi que la granularité des états plutôt que d'avoir une statechart minimale qui est très éloigné du contrôle.

Cependant la date du rendu approchant et n'ayant pas réussi à faire fonctionner cette solution nous avons décidé de nous pencher sur une autre solution.Pour résoudre nos problèmes, nous avons décidé d'enlever l'état parallèle et de développer en détail ce qui se passe dans DodgeObstacle. Alors nous avons réussi à avoir la statechart suivante :

Notre version finale:



Nous avons rajouté deux nouveaux states, DodgeRightObstacle et DodgeLeftObstacle pour que le robot réagit différemment si l'obstacle se trouve à droite ou à gauche, en utilisant les détecteurs qu'il possède.

Pour éviter le mur, nous avons rajouté l'état DodgeVirtualWall pour que le robot fasse un tour complet s'il y a un mur à côté de lui.

Nous avons aussi amélioré notre code, et nous avons rajouté des Observers pour réagir aux événements générés par notre state machine.

La détection n’est donc plus gérer dans la statechart mais dans le code, c’est le code qui va appeler les mesures de distances des détecteurs en permanence plutôt que dans la version précédente où l’on faisait appelle à un état parallèle. Les avantages de cette statechart est qu’elle est déjà plus granulaire que ce qui a été décrit en deuxième solution, et qu'elle ne nécessite pas de synchronisation, mais elle n’utilise pas la bonne propriété que nous avons évoquée dans la première solution. Les inconvénients réside dans la difficulté à rajouter des extensions, en effet tout ce qui sera lié à la détection sera donc obligatoirement dans le code.De plus le seul moyen que nous avons de nous assurer que le robot se comporte correctement est dans les transitions possibles définies depuis un état ce qui peut rendre le contrôle exhaustif assez délicat à tester.

III/ V&V

Nous avons réalisé deux sets de tests, le test3 qui reprend notre premier statechart ainsi que le test4 tous deux évaluent des propriétés de vérification et sont des tests de liveness.

Test3 :

(Ici dodgingObstacle correspond à Obstacle Detected de notre statechart)

Nous avons essayé de vérifier si le problème que nous avions avec le robot venez de la statechart ou bien du code. Pour cela nous avons posé 3 questions de vérification :

Quand je rentre dans dodgeObstacle inévitablement je rentre dans Forward

Quand je sors de dodgeObstacle éventuellement je rentre dans Forward et je rentre dans checkObstacle

Quand je sors de dodgingObstacle inévitablement je rentre dans Forward

A ces trois questions nous avons obtenu la réponse TRUE. Ce qui indique que le problème que nous avons ne vient pas de notre statechart.

Test4 :

Pour notre dernière statechart nous avons vérifié le comportement de notre statechart de la même manière que précédemment.

Quand je rentre dans DodgeObstacle éventuellement je rentre dans Forward

Quand je rentre dans dodgeVirtualWall éventuellement je rentre dans Forward

Quand je rentre dans Forward éventuellement je rentre dans DodgeVirtualWall

Quand je rentre dans Forward éventuellement je rentre dans DodgeObstacle

Bien que ces questions ne soient pas forcément les plus importantes, elles nous assurent le comportement du robot.

Les statechart que nous avons utilisés pour nos tests ne contiennent qu'une partie de ce que nous avons fait, elles contiennent la partie correspondant au MVP. Donc seulement la partie qui fait esquiver les obstacles et qui fait avancer le robot. La plus grande difficulté était de s'assurer que ce que nous demandions correspond bien à ce que nous voulions demander.

IV/ Prise de recul

Nous avons pu apprécier l'utilité d'une statechart, notamment sur l'organisation du contrôle. Au premier abord un tel projet paraît impossible niveau contrôle mais à l'aide de la statechart nous avons pu quand même avoir une idée assez précise de la manière de procéder, de la manière de réaliser des tests de validation et de vérification. La statechart nous a permis de nous réconforter sur ce que nous traitions, en nous assurant que nous n'avions pas rater un ou plusieurs cas particuliers (ce qui aurait surement été le cas avec des suites de if ou bien avec des switch cases). Nous aurions dû nous concentrer au début uniquement sur la partie MVP plutôt que d'essayer de construire un MVP qui puisse facilement s'étendre à des options supplémentaires.