Premiers pas en robotique

avec le robot

Robot Thymio

et l'environnement

Aseba/VPL

Moti Ben-Ari et autres contributeurs

voir `authors.txt` pour plus de détails

Version 1.4.1 pour Aseba 1.4

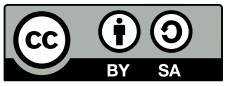


Table des matières

Préface

Qu'est-ce qu'un robot ?

Imaginez-vous sur votre vélo, pédalant sur la route, lorsque vous vous trouvez face à une colline. Vous décidez donc de pédaler plus vite pour ne pas perdre trop de vitesse. Une fois en haut, vous vous trouvez face à une descente plutôt raide. Vous allez donc freiner pour éviter de prendre trop de vitesse et de perdre le contrôle de votre vélo. Lorsque vous êtes sur votre vélo, vos yeux sont vos *capteurs* qui mesure ce qu'il se passe dans le monde. Lorsque ces capteurs — vos yeux – détectent un *événement* tel qu'une courbe de la route, vous effectuez une *action*, telle que bouger le guidon à gauche ou à droite.

Dans une voiture, nous pouvons trouver de nombreux *capteurs*. Le tachymètre, ce cadran à aiguille ou digital derrière le volant, vous permet de savoir à quelle vitesse votre voiture avance. Si vous remarquez que vous allez trop vite par rapport à la limitation, vous allez freiner et, au contraire, vous accélérerez si vous vous trouvez en dessous de la limite. Un autre cadran vous indique la quantité d'essence se trouvant dans votre réservoir. Si vous voyez qu'il n'y en a presque plus, vous déciderez d'entreprendre l'action d'aller remplir votre réservoir.

Autant sur votre vélo que dans votre voiture, vous recevez des informations, des données, depuis les capteurs et vous décidez d'entreprendre des actions vis-à-vis de ces données. Un *robot* est un système dans lequel le processus est effectué par un système informatique, en général sans la participation d'un être humain.

Le robot Thymio et l'environnement Aseba VPL

Thymio est un petit robot conçu à but éducatif (Figure 1.1). Il comprend différents capteurs qui peuvent mesurer la lumière, les distances, la température, etc. Il possède également des boutons tactiles, il peut se rendre compte qu'on lui donne une petite tape et peut entendre quelqu'un qui frappe dans ses mains. Il possède deux roues, chacune reliée à un moteur, lui permettant de se déplacer sur une table, ou sur le sol. Il peut encore s'illuminer de toutes les couleurs et jouer de la musique !

Dans ce document, le nom Thymio est utilisé pour désigner le robot Thymio II.

Aseba est un environnement de programmation pour petits robots comme le Thymio. VPL est un composant d'Aseba qui permet à toutes et tous de programmer *graphiquement* Thymio en utilisant des blocs événement et action très simples d'emploi.

Vue d'ensemble du tutoriel

Voici les sujets et les listes des blocs évènement et d'action qui seront traités dans chaque chapitre. Je vous suggère de commencer par les chapitres sur le mode de base de VPL. Par

la suite, vous pourrez suivre le tutoriel en mode avancé ou vous lancer dans quelques uns des projets présentés. Les casse-têtes de Parson peuvent être tentés à tout moment si vous voulez tester vos connaissances de VPL. Lisez le Chapitre 21 lorsque vous voudrez quitter VPL pour utiliser l'environnement plus avancé Aseba Studio. Des résumés de référence sont aussi à votre disposition en annexes.

Première partie : le tutoriel

Les Chapitres 1 et 2 sont une introduction essentielle du robot, de l'environnement VPL et de son principal concept de programmation : les paires événement-actions.

Événements : boutons **Actions** : couleurs du haut et du bas



Les Chapitres 3 à 5 présentent les événements, actions et algorithmes nécessaires à la construction de robots mobiles autonomes et qui sont au centre de toute activité avec Thymio et VPL.

Événements : boutons, capteurs avants, capteurs inférieurs. **Actions** : moteurs




Le Chapitre 6 décrit certaines fonctionnalités amusantes du robot mais qui ne sont pas nécessaires : les sons et les chocs.

Événements : petite tape, frappe des mains **Actions** : musique, couleurs du haut et du bas





> **Mode avancé**
>
> VPL contient un mode de base permettant l'utilisation d'événements et actions faciles à maîtriser pour un débutant. Le mode avancé de VPL offre plus d'événements et d'actions mais requière plus d'expérience. Les fonctionnalités du mode avancé sont présentées à partir du Chapitre 7.

Le Chapitre 7 explique les événements minutés. Il existe une action qui enclanche un minuteur. Un événement est ensuite déclenché quand le minuteur est écoulé.

Événements : minuteur écoulé **Actions** : démarer le minuteur.


Les Chapitres 8 et 9 expliquent le concept de machines à états qui permettent au robot de agir différemment selon leur état. Les états permettent aussi des opérations arithmétiques élémentaires comme compter.

Événements : état associé à un événement **Actions** : changer d'état


Le Chapitre 10 décrit l'utilisation des accéléromètres du robot Thymio.

Événements : Événements accéléromètre


Deuxième partie : les casse-têtes de Parson

Le Chapitre 11 présente les casse-têtes de Parson, des exercices qui vous permettent d'évaluer vos connaissances de VPL.

Troisième partie : les projets

Les Chapitres 12 à 20 proposent des projets que vous pouvez réaliser vous-mêmes. Le code source VPL est disponible dans l'archive mais je vous suggèrerais d'avancer seul avant de comparer avec le corrigé.

Quatrième partie : de la programmation visuelle à la programmation textuelle

Le Chapitre 21 ouvre la voie pour la suite : l'utilisation de l'environnement textuel Studio. Avec celui-ci, il est possible d'aller bien au-delà des possibilités offertes par VPL.

Cinquième partie : les annexes

L'Annexe A contient une description de l'interface utilisateur — les boutons sur la barre d'outils.

L'Annexe B est une liste des blocs événement et action pour les modes de base et avancé.

L'Annexe C donne quelques propositions destinées aux professeurs et directeurs de projets d'étudiants. La première section offre des pistes pour stimuler l'exploration et l'expérimentation. La section suivante donne quelques bonnes habitudes de programmation. La section finale dresse une liste de quelques pièges courants et comment les éviter.

L'Annexe D discute différentes techniques d'utilisation des *sliders* pour les blocs capteurs et moteurs.


Cartes de référence

Vous trouverez peut-être utile d'imprimer une ou les deux cartes de référence VPL. Vous les trouverez dans le même fichier zip que ce document, ou alors sur https://www.thymio.org/fr:visualprogramming.

— Une page simple contient un résumé des blocs événement et action.

— Une page recto-verso qui peut être pliée pour former une carte pratique contient un résumé de l'interface VPL, des blocs événement et action et des exemples de programmes.

Installation d'Aseba

Pour installer Aseba, y compris VPL, rendez-vous sur https://www.thymio.org/fr:start et cliquez sur l'icône correspondant à votre système (Windows, Mac OS etc.). Suivez les instructions pour télécharger et installer l'application. L'insallation d'Aseba contient à la fois l'environnement de développement VPL et Studio (voir Chapitre 21).

Versions 1.3 et 1.4

La liste suivante est destinée aux lecteurs ayant déjà une certaine expérience avec VPL dans sa version précédente, Aseba 1.3 ; elle énumère les différences entre cette version et la 1.4. Si vous n'avez jamais utilisé VPL auparavant, vous pouvez sauter cette section.

Changements apportés aux blocs événement et action

- L'apparence graphique des boutons des blocs a été modifiée, principalement dans le but d'intégrer de nouvelles fonctionnalités.
- Dans la 1.3, une case *rouge* dans un bloc événement capteurs horizontaux déclanchait un événement lorsque le capteur détectait un objet, alors qu'une case *blanche* déclanchait un événement lorsqu'il n'y avait pas d'objet en face du capteur. Dans la 1.4, une case *blanche* déclanche un événement quand le capteur mesure beaucoup de lumière réfléchie, alors qu'une case *noire* déclanche un événement lorsque le capteur mesure peu ou pas de lumière réfléchie parce qu'il n'y a pas d'objet devant le capteur (pages 18, 20). Les capteurs du bas utilisent aussi les couleurs noir et blanc, au lieu de rouge et blanc, mais le fonctionnement dans la 1.4 est identique à la 1.3, avec le noir au lieu du rouge.
- En mode avancé, les seuils des capteurs peuvent être modifiés (page 46).
- En mode avancé, un événement permet de créer des événements accéléromètre en indiquant des valeurs pour les accéléromètres avant/arrière ou gauche/droite (page 46).

Modifications de l'interface utilisateur

- Des actions multiples associées au même événement (page 16).
- Les blocs et les paires événement-actions peuvent être copiées (page 12).
- Les captures d'écran des programmes VPL peuvent être exportés en plusieurs formats (page 91).
- Les boutons annuler/rétablir ont été ajoutés (page 82).
- Le bouton Lancer clignote en vert lorsque le programme a été modifié (page 12).
- Il n'est désormais plus possible de modifier la palette de couleurs de VPL.

Première partie

Tutoriel

Chapitre 1

Votre premier projet de robotique

Faire connaissance avec Thymio

La Figure 1.1 montre l'avant et le dessus de Thymio. Vous pouvez voir le bouton central rond (A), entouré de quatre boutons triangulaires (B). Ce sont des boutons tactiles, un simple effleurement suffit à les activer. Juste derrière ces boutons se trouve un indicateur en forme de pile (C). Une, deux ou trois petites barres de lumière verte affichent l'état de charge du robot. Sur l'arrière du robot, nous voyons les lumières du haut (D), allumées en rouge sur cette photo. Il y a des lumières similaires sur le bas du robot (voir Figure 3.1). Enfin, les petits rectangles noirs à l'avant du robot (E) sont des capteurs infrarouge de distance, vous en saurez plus dans le Chapitre 4.

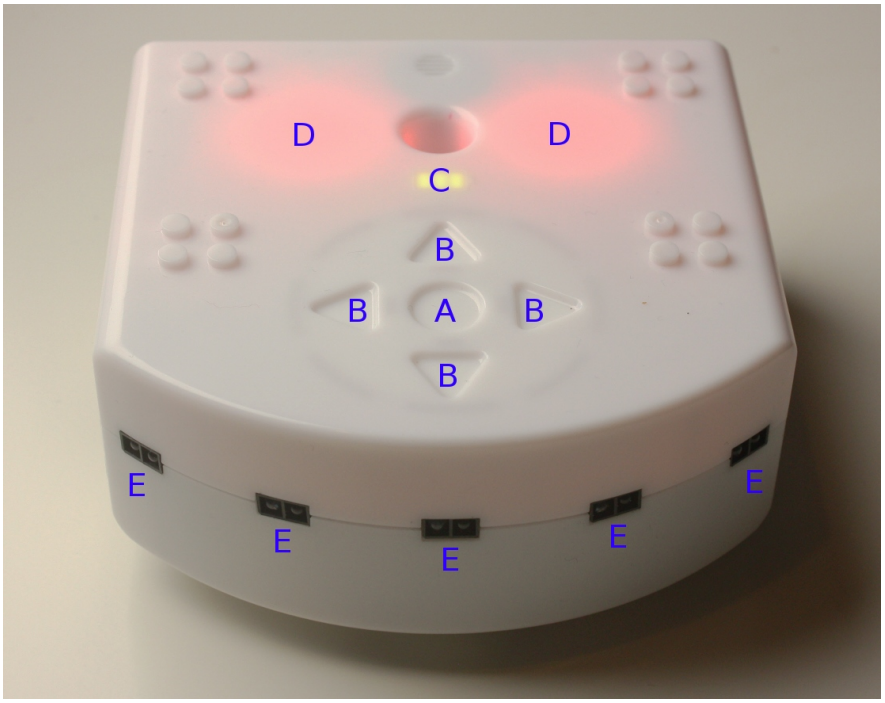


FIGURE 1.1 – L'avant et le dessus de Thymio

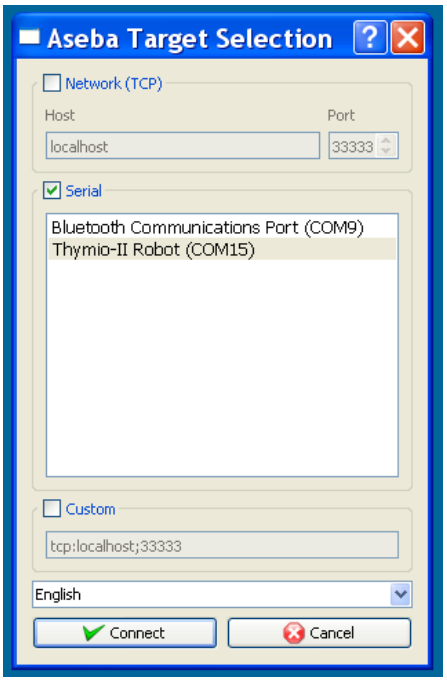


FIGURE 1.2 – Se connecter à Thymio via le port série (USB)

Connecter le robot, démarrer Aseba et lancer VPL

Pour commencer, connectez Thymio à votre ordinateur à l'aide du câble USB fourni avec le robot. Si la connection est réussie, le robot jouera quelques notes. S'il est éteint, touchez simplement son bouton central pendant quelques secondes jusqu'à entendre quelques notes. Lancez VPL en double-cliquant sur l'icône de votre ordinateur.

Petites images

Quand le texte contient une petite image, la même image en plus grand apparaît aussi dans la marge.

Il se peut que VPL se connecte directement à votre robot. Si ce n'est pas le cas, la fenêtre montrée dans la Figure 1.2 devrait apparaître. Cochez la case **Port série**, cliquez sur **Thymio-II Robot**, sélectionnez **Français** et cliquez sur **Connecter**. En fonction de la configuration de votre ordinateur et de votre système d'exploitation, il peut y avoir plusieurs entrées dans la liste des ports séries et le texte à côté de **Thymio-II Robot** peut différer de la Figure 1.2.

Truc

Il est aussi possible d'accéder à VPL depuis Aseba Studio, l'environement de programmation textuelle, à travers le *plugin* VPL qui se trouve dans la zone *Outils* en bas à gauche de l'écran.

L'interface VPL

L'interface VPL est illustrée ci-dessous. Elle est composée de six zones :

1. Une barre d'outils avec les boutons pour créer un nouveau programme, en ouvrir un existant, sauvegarder, lancer le programme, etc.
2. La zone de programmation pour construire le programme qui contrôlera Thymio.
3. Une zone de messages qui donne les messages d'erreur lorsque le programme n'est pas construit correctement.
4. Les blocs d'événement disponibles pour construire votre programme.
5. Les blocs d'action disponibles pour construire votre programme.
6. La traduction du programme en AESL, le langage textuel d'Aseba.

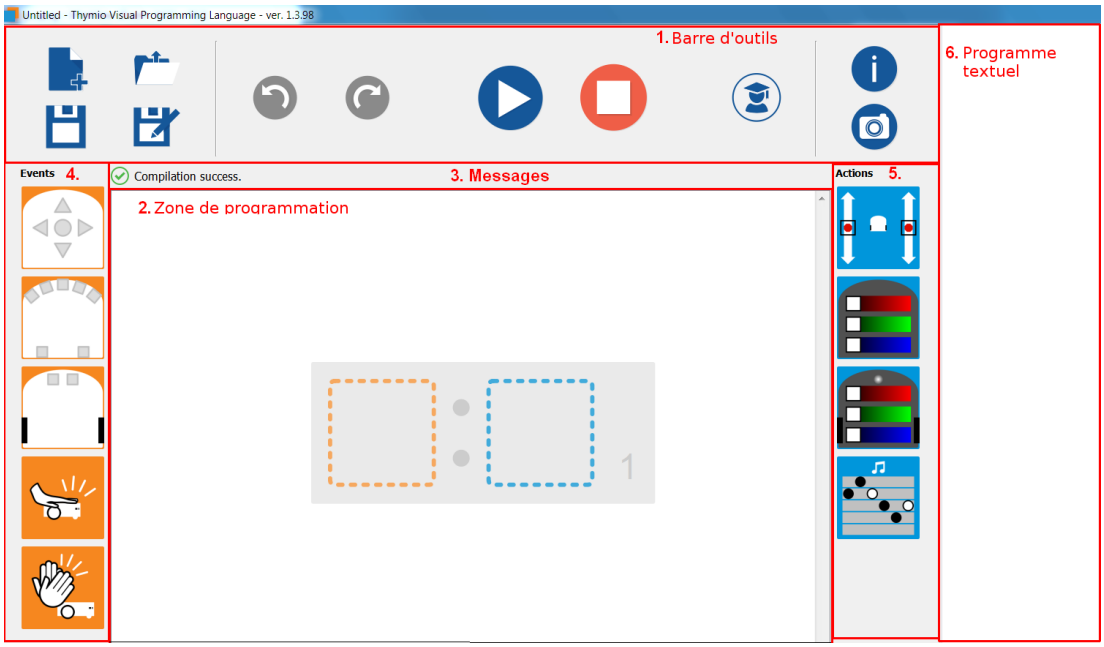


FIGURE 1.3 – La fenêtre de VPL

> **Pour aller plus loin**
>
> Dès que vous créez un programme en utilisant VPL, la traduction du programme dans le langage de programmation textuel AESL apparaît dans la partie droite de la fenêtre. C'est en fait ce code AESL qui est interprêté par le robot. Si vous êtez curieux et que vous désirez comprendre ce langage, vous pouvez lire le Chapitre 21 qui explique ces traductions. Vous pouvez ensuite vous rendre sur https://aseba.wikidot.com/fr:asebausermanual pour les références et en apprendre plus sur AESL et son environnement Studio.

Écrire un programme

Quand vous démarrez VPL, une zone de programmation vide est affichée.

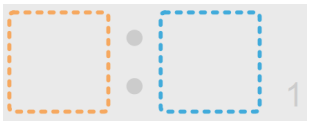
Si, après avoir construit un bout de programme, vous voulez effacer le contenu de la zone de programmation, vous pouvez cliquer sur  (**Nouveau**).

Un programme dans VPL consiste en des *paires événement-actions*, chacune construite en mettant ensemble un bloc événement et un ou plusieurs bloc action. Par exemple, la paire :

allume la lumière du haut du Thymio en rouge lorsque l'on touche le bouton avant sur le robot.

> **Signification d'une paire événement-actions**
>
> Lorsque l'événement se produit, le robot exécute les actions associées.

Dans la zone de programmation se trouve initialement une paire événement-actions vide :



Pour placer un bloc dans le programme depuis une des colonnes (zones 4 et 5 de Figure 1.3), cliquez puis gardez enfoncé le bouton gauche de la souris. Glissez ensuite le bloc jusqu'à un carré traitillé. Lorsque le bloc est au-dessus du carré, relâchez le bouton de la souris pour déposer le bloc à sa place.

> **Information importante**
>
> On appelle *glisser-déposer* (ou *drag-and-drop* en anglais) la technique qui vient d'être décrite. Elle est fréquemment utilisée dans les interfaces de programmes.

Commencez par amener le bloc évenement boutons  sur le carré vide de gauche. Vous recevrez alors un message vous invitant à ajouter un bloc action. Amenez le bloc action couleur du haut  sur le carré de droite. Et voilà ! Vous avez construit une paire événement-actions !

Il nous faut maintenant modifier l'événement et l'action pour qu'ils fassent ce que l'on veut. Pour l'événement, cliquez sur le bouton avant (le triangle supérieur) ; il va devenir rouge :

Cela signifie qu'un événement se produira lorsque le bouton *avant* de Thymio sera touché.

Le bloc action couleur contient trois *sliders* — des barres de couleur avec un carré blanc. Chaque barre règle une des trois couleurs primaires rouge, vert et bleu. Si vous déplacez un de ces carrés blanc vers la droite et revenez ensuite en arrière, vous vous apercevrez que le

fond du bloc change de couleur. En mélangeant ces trois couleurs primaires (rouge, vert et bleu), toutes les couleurs peuvent être créées. Déplacez le *slider* rouge jusqu'à ce que le carré soit tout à droite et déplacez les *sliders* vert et bleu tout à gauche. La couleur sera toute rouge sans composante bleu ou vert :

Sauvegarder le programme

Avant de lancer votre programme, sauvegardez-le sur votre ordinateur. Cliquez sur l'icône (**Sauvegarder**) de la barre d'outils. Vous devrez choisir un nom pour votre programme, par exemple **afficher-rouge**. Choisissez l'endroit où vous voulez sauvegarder le programme, sur le bureau par exemple, et cliquez sur **Sauvegarder**.

> **Sauvegardes fréquentes**
> Lorsque vous modifiez un programme, cliquez souvent le bouton **Sauvegarder** pour éviter de ne perdre votre travail si un problème devait survenir avec votre ordinateur.

Lancer le programme

Pour lancer le programme, cliquez sur (**Lancer**) dans la barre d'outils. Essayez maintenant d'appuyer sur le bouton avant de Thymio, il devrait s'être allumer en rouge !

> **Félicitations !**
> Vous avez créé et exécuté votre premier programme. Voici ce qu'il fait :
> **Lorsque l'on touche le bouton avant de Thymio, il devient rouge.**

Si vous voulez arrêter le programme VPL, cliquez sur (**Stop**). Ceci peut être très utile lorsque par exemple vous exécutez un programme qui fait avancer le robot mais que vous avez oublié d'ajouter une paire événement-actions pour arrêter les moteurs.

Éteindre le robot

Lorsque vous avez terminé d'utiliser Thymio, vous pouvez l'éteindre en touchant son bouton central et en gardant le contact quelques secondes. Vous entendrez quelques notes et Thymio s'arrêtera. Tant que le robot est connecté à un ordinateur, sa batterie continue à se recharger. Une petite lumière rouge à l'arrière du robot, juste à côté du câble USB, permet de savoir s'il est rechargé ou pas. La lumière passe du rouge au bleu pour indiquer qu'il est complètement

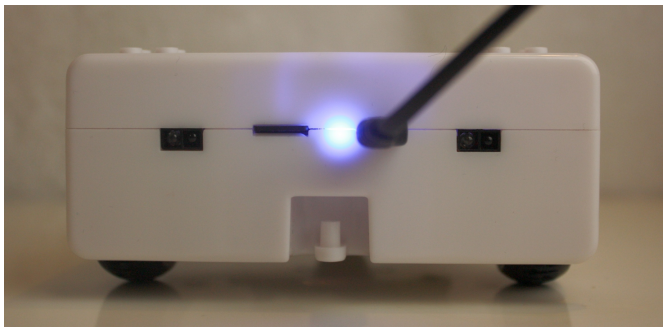


FIGURE 1.4 – L'arrière de Thymio avec le câble microUSB et le témoin de charge

rechargé, comme sur la Figure 1.4. Vous pouvez déconnecter le cable quand vous n'utilisez pas le robot.

Truc

Si vous voulez charger le robot plus vite, vous pouvez le connecter à une prise murale avec un chargeur pour téléphone portable fournissant une prise micro-USB.

Si le cable USB se déconnecte durant la programmation, VPL attendra une reconnection. Vérifiez les deux côtés du cable, débranchez et rebranchez le câble, et regarder si VPL fonctionne. Si vous avez un problème, vous pouvez toujours fermer VPL, reconnecter le robot et réouvrir VPL.

Modifier un programme

- Pour effacer une paire événement-actions, cliquez sur ×, en haut à droite de la paire.
- Pour ajouter une paire événement-actions, cliquez sur +, disponible en dessous de chaque paire.
- Pour déplacer une paire événement-actions, maintenez le bouton gauche de la souris enfoncé sur une paire et glissez la à l'endroit désiré.
- Pour copier une paire événement-actions vers un deuxième endroit dans le programme, appuyez et maintenez pressé la touche **Ctrl** et utilisez votre souris pour glisser et déposer la paire à l'emplacement désiré. [1]

Le bouton Lancer clignote

Lorsque vous modifiez un programme, le bouton **Lancer** clignote en bleu et vert pour vous rappeler qu'il faut cliquer sur le bouton pour charger le programme modifié vers le Thymio.

1. Sur Mac OS, la touche **Command** remplace la touche **Ctrl**.

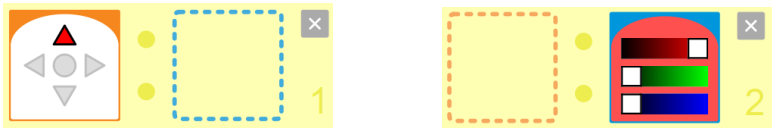
Si vous voulez tester une modification sans perdre votre programme actuel, vous pouvez créer une copie du programme actuel en cliquant sur (**Sauvegarder sous**) et en entrant un nouveau nom de fichier.

Ouvrir un programme existant

Si vous voulez continuer un programme que vous aviez commencé précédemment, il suffit de l'ouvrir avec l'interface VPL pour le modifier ou l'améliorer. Cliquez sur l'icône (**Ouvrir**) et sélectionnez le fichier que vous voulez ouvrir, par exemple **afficher-rouge**. Les paires événement-actions du programme vont être affichées dans la zone de programmation, et vous pourrez continuer à les modifier.

La paire événement-actions actuelle

Lorsque vous cliquez sur une paire événement-actions, celle-ci s'affiche avec un fond jaune. Il en va de même lorsque vous ajoutez un bloc événement ou action dans une paire vide.



Le carré gauche de couleur dorée est l'espace réservé à l'événement ; le carré bleu à droite sert d'emplacement à la première (ou unique) action. La pair au fond jaune est appelée la paire actuelle.

> **Insérer un bloc rapidement**
> Si vous cliquez sur un bloc événement ou action, il sera automatiquement placé dans la zone de programmation dans la paire événement-actions sélectionnée.

> **La barre d'outils VPL**
> L'Annexe A contient une description de tous les boutons de la barre d'outils VPL. Consultez la de temps en temps jusqu'à ce que vous ayez appris à les utiliser.

Chapitre 2

Changer les couleurs

Colorer Thymio

Créons un programme qui affiche deux couleurs différentes sur le dessus de Thymio lorsque les boutons avant ou arrière sont touchés, et deux autres couleurs sur le dessous et les côtés de Thymio lorsque les boutons gauche ou droite sont touchés.

Programme : **colors.aesl**

Nous avons besoin de quatre paires événement-actions. Il y a quatre événements — toucher l'un des quatre boutons — et une action couleur est associée avec chaque événement. Notez la différence entre le bloc  et le bloc  . Le premier des deux change la couleur affichée sur le dessus de Thymio alors que le deuxième change la couleur affichée sur le dessous et sur les côtés. Le deuxième bloc a deux marques noires qui représentent les roues du robot et un point blanc qui représente le point d'appui à l'avant du robot (Figure 3.1).

Ce programme est illustré sur la Figure 2.1.

Quelles couleurs seront affichées ? Pour les premières trois actions, le *slider* d'une couleur a été glissé tout à droite alors que les autres sont restés à gauche. Ces actions affichent donc respectivement purement du rouge, du bleu et du vert. L'action associée avec le bouton gauche mixe du rouge et du vert, ce qui produit donc du jaune. Vous pouvez voir que le fond du bloc change de couleur quand on déplace les *sliders*, ce qui vous montre de quelle couleur sera Thymio !

Lancez le programme  et touchez les boutons pour changer la couleur du robot. La Figure 1.1 montre Thymio allumé en rouge sur le dessus et la Figure 3.1 montre Thymio allumé en vert sur le dessous.

Exercice 2.1

Expérimentez avec les *sliders* pour voir quelles couleurs peuvent être affichées.

Information

En mixant ensemble du rouge, du vert et du bleu, vous pouvez créer n'importe quelle couleur (Figure 2.2) !

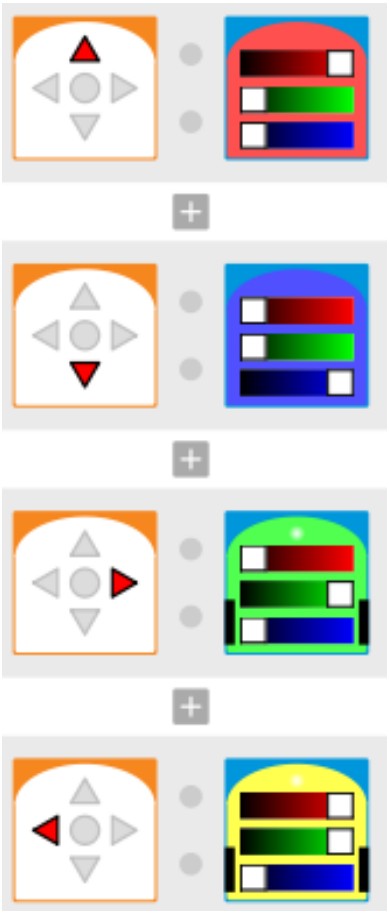


FIGURE 2.1 – Changer de couleur quand un bouton est touché

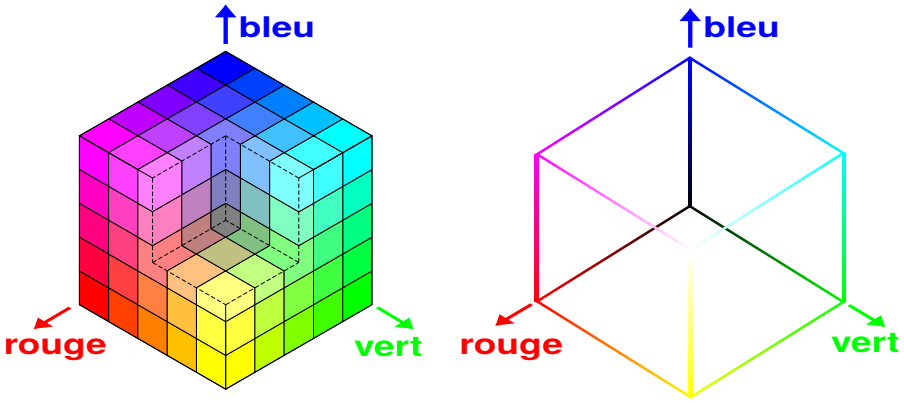
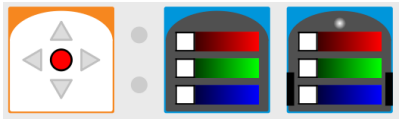


FIGURE 2.2 – Le cube des couleurs rouge-vert-bleu (RVB)

Plusieurs actions associées à un même événement

Modifions maintenant le programme pour que les lumières s'éteignent lorsque le bouton central est touché. Deux actions doivent avoir lieu lors d'un même événement — le bouton du milieu est touché. Il est possible d'associer *deux* actions à un événement dans une paire événement-actions. Après avoir inséré l'événement et la première action (l'action couleur du haut), un nouveau cadre gris apparaîtra à droite de l'action :

Vous pouvez maintenant glisser-déposer l'action couleur du bas dans ce cadre. Vous obtiendrez une paire avec un événement et deux actions :



Programme : **colors-multiple.aesl**

N'oubliez pas de lancer le programme en cliquant sur l'icône ▶. À l'avenir, il sera implicite qu'il vous faudra lancer chaque programme en cliquant sur cet icône, nous ne vous le dirons plus.

Les règles des paires événement-actions

- Lorsqu'un programme est lancé, toutes les paires événement-actions sont actives.
- Il est possible d'avoir plusieurs paires événement-actions avec le même événement mais il faut que les paramètres associés soient différents. Vous pouvez par exemple avoir plusieurs paires avec l'événement bouton à condition que les combinaisons de boutons soient différentes entre les différents blocs événement.
- Si les événements sont absoluement identiques dans plusieurs paires, VPL vous indiquera qu'il y a une erreur (zone 3 dans la Figure 1.3). Vous ne pourrez pas lancer le programme tant qu'il y a des erreurs.

Chapitre 3

Thymio en mouvement

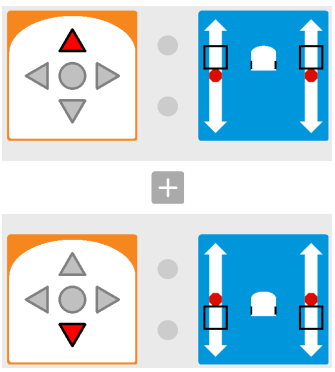
En avant, en arrière

Thymio a deux moteurs, un sur chaque roue. Ils peuvent tourner dans les deux sens, permettant à Thymio d'avancer, de reculer et de tourner. Commençons par un petit programme qui vous apprendra à contrôler les moteurs.

Le bloc action moteurs représente Thymio entouré de deux *sliders*. Chaque *slider* contrôle un moteur. En les glissant vers l'avant, Thymio avancera, à l'inverse, en les glissant vers l'arrière, Thymio reculera. Pour arrêter les moteurs, il suffit de glisser les *sliders* au centre des barres. Écrivons un programme qui fasse avancer Thymio lorsque l'on touche le bouton avant et qui le fasse reculer lorsque l'on touche le bouton arrière.

Programme : **moving.aesl**

Nous allons avoir besoin de deux paires événement-actions :



Amenez les blocs événement et action dans la zone de programmation et ajustez les *sliders* des moteurs identiquement à droite et à gauche, à la moitié vers le haut pour avancer et à la moitié vers le bas pour reculer.

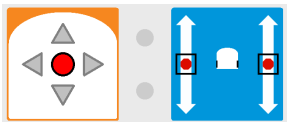
Lancez maintenant le programme et touchez les boutons avant et arrière du robot pour faire avancer et reculer Thymio !

Arrête-toi !

À l'aide ! Thymio ne veut plus s'arrêter !

Cliquez sur le bouton **(Arrêter)** pour arrêter Thymio.

Nous allons corriger ce problème en ajoutant une paire événement-actions qui va arrêter les moteurs quand le bouton central est touché :



Lorsque vous ajoutez un bloc moteur dans le programme, il est réglé avec les *sliders* en position médiane, de sorte que les moteurs s'arrêtent.

Ne tombe pas de la table

Si Thymio se trouve sur le sol, au pire, il rentrera dans un mur, mais s'il est sur une table, il risque de tomber ! Nous allons créer un petit programme qui lui permettra de s'arrêter s'il arrive au bord de la table.

> **Attention !**
>
> Si Thymio roule sur une table, tenez-vous toujours prêt(e) à le rattraper s'il arrive près du bord de la table !

Tournez Thymio sur son dos. Vous verrez deux petits rectangles noirs qui contiennent des éléments optiques : on peut les voir sur le haut de la Figure 3.1. Ce sont les *capteurs du sol*. Ils envoient une impulsion de lumière infrarouge et mesurent la quantité de lumière qui leur est réfléchie. Si Thymio est posé sur une table de couleur claire, beaucoup de lumière sera réfléchie, alors que s'il dépasse le bord de la table, peu de lumière sera réfléchie. Nous allons donc utiliser ces capteurs pour dire à Thymio de s'arrêter lorsqu'il arrive au bord de la table.

> **Truc**
>
> Utilisez une table de couleur claire ou fixez des feuilles de papier blanc sur la table. Évitez les tables en verre transparent, elles ne réfléchiront probablement pas la lumière et Thymio croira qu'il n'est pas sur une table !

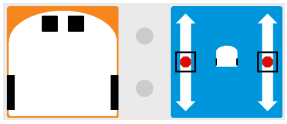
Glisser-déposer le bloc capteurs du sol sur la zone de programmation pour commencer. Les deux petits carrés gris représentent les détecteurs de sols. En cliquant sur ces carrés, ils passent de gris à rouge, à noir puis à nouveau à gris. Pour ce bloc, ces couleurs signifient :

- **Gris** : Le détecteur n'est pas utilisé.
- **Blanc** : L'action associée est déclenchée s'il y a beaucoup de lumière réfléchie. À côté du carré blanc, il y a un petit point rouge ; ce point correspond à la petite lumière rouge visible à côté de chaque capteur et qui s'allume lorsque le capteur détecte quelque chose. [1]
- **Noir** : L'action associée est déclenchée s'il y a peu de lumière réfléchie.

1. Le carré blanc a un bord rouge pour vous rappeler que l'événement sera déclenché lorsque les lumières à côtés des capteurs seront elles-mêmes allumées.

FIGURE 3.1 – Le dessous de Thymio avec ses détecteurs de sols

Pour que Thymio s'arrête au bord de la table (lorsqu'il y a peu de lumière réfléchie) cliquez sur les deux carrés jusqu'à ce qu'ils soient noirs. Créez la paire événement-action suivante :



Placez Thymio près d'un bord d'une table de façon à ce qu'il soit face au bord et touchez le bouton avant. Thymio devrait avancer jusqu'au bord et s'y arrêter.

Exercice 3.1

Modifiez la vitesse du bloc action moteurs de Thymio. À sa vitesse maximale, Thymio est-il toujours capable de s'arrêter avant le bord de la table ? Si non, à partir de quelle vitesse le robot ne peut plus s'arrêter ? Pouvez-vous empêcher le robot de tomber s'il va en arrière ?

Attention !

Lorsque j'ai lancé moi le programme, le robot *est* tombé. La raison était que mon bureau avait un bord arrondi ; d'ici à ce que le robot ait détecté un faible niveau de lumière, il n'était déjà plus stable et a basculé. Si vous voulez arrêter Thymio un peu avant le bord de la table, vous pouvez placer une feuille noir, ou du ruban adhésif noir, là où vous souhaitez qu'il s'arrête !

Chapitre 4

Un robot de compagnie

Un *robot autonome* adopte un comportement spécifique en fonction de la situation dans laquelle il se trouve. Il arrive à réagir grâce au *feedback*, littéralement « l'information en retour » en anglais. Il faut donc que le robot puisse « voir » le monde qui l'entoure pour pouvoir y réagir.

Thymio vous obéit

Nous allons programmer Thymio pour qu'il vous obéisse : le robot restera sur place sans bouger, mais quand il détectera votre main devant lui, il bougera dans sa direction.

Programme : **obeys.aesl**

Thymio a cinq capteurs de proximité horizontaux à l'avant et deux à l'arrière. Ils sont similaires à ceux qui se trouve sous Thymio et que nous avons utilisés au Chapitre 3. Avancez votre main en direction des capteurs à l'avant de Thymio et vous verrez une lumière rouge apparaître à côté du capteur qui vous aura détecté, comme sur la Figure 4.1.

Le bloc  sert à détecter si quelque chose est proche d'un des capteurs horizontaux avants et arrières de Thymio. Dans les deux cas, un événement est déclenché. Les petits carrés gris (cinq sur l'avant et deux sur l'arrière) sont utilisés comme pour les capteurs situés sous Thymio. En cliquant dessus, ils passent de gris à blanc, à noir et à nouveau à gris. Les significations des couleurs sont :

— **Gris** : Le capteur n'est pas utilisé.

— **Blanc** : L'action associée est déclenchée s'il y a beaucoup de lumière réfléchie. Le carré blanc a un bord rouge pour vous rappeler que l'événement sera déclenché lorsque les lumières à côtés des capteurs seront elles-mêmes allumées.

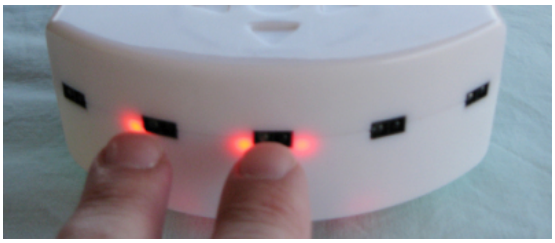


FIGURE 4.1 – L'avant de Thymio. Deux doigts sont détectés par les capteurs avant.

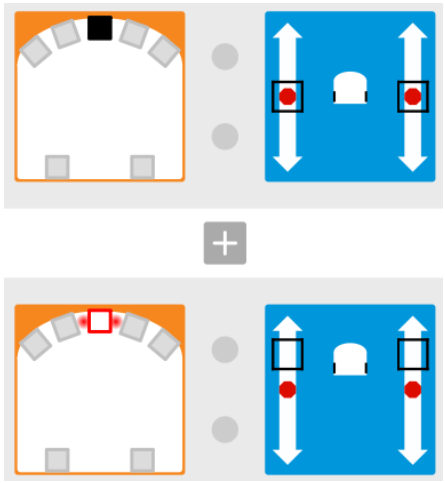


FIGURE 4.2 – Thymio avance vers votre main

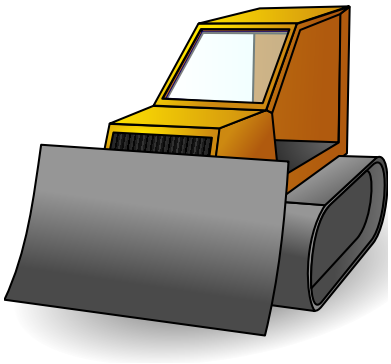


FIGURE 4.3 – Un bulldozer à chenilles

— **Noir** : L'action associée est déclenchée s'il y a peu de lumière réfléchie.

Si vous souhaitez qu'un événement se produise quand un objet est proche d'un capteur, utilisez les carrés blancs car l'objet réfléchira beacoup de lumière. Si vous souhaitez au contraire qu'un événement se produise quand aucun objet ne se trouve à proximité d'un capteur, utilisez un carré noir car peu de lumière sera réfléchie.

Pour construire le comportement, il nous faut deux paires événement-actions, (Figure 4.2). Dans ce programme, vous voyez que dans la première paire, le carré central est noir et l'action associée est que les moteurs sont arrêtés. Ainsi, lorsque le robot ne voit rien, il ne bougera pas ; et s'il bougeait, il s'arrêtera. Dans la deuxième paire, le carré central est rouge et les *sliders* du bloc action moteurs sont glissés vers le haut. Ainsi, lorsque vous amenez votre main près de l'avant du robot, un événement se produit qui fait tourner les deux moteurs assez vite et fait avancer le robot en avant.

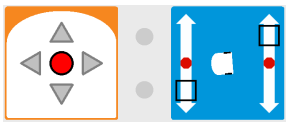
Faire tourner Thymio

Thymio n'a pas un volant comme une voiture ou un guidon comme un vélo. Comment tourne-t-il donc ? Pour tourner, le robot utilise une *direction différentielle*, ou *differential drive* en anglais, qui est utilisée par de nombreux véhicules à chenilles comme le bulldozer sur la Figure 4.3. À la place de tourner un guidon dans la direction désirée, les chenilles ou roues gauches et droites sont commandées par des moteurs individuels à des vitesses *différentes*. Si la roue droite tourne plus vite que la roue gauche, alors le véhicule tournera à gauche, tandis que si sa roue gauche tourne plus vite que sa roue droite, il tournera à droite.

La direction différentielle pour le Thymio est réalisée en réglant les différents *sliders* d'un bloc action moteurs à des endroits différents, de sorte que les roues de Thymio ne tourneront pas à la même vitesse, ce qui fera tourner le robot. Plus la différence de vitesse est élevée, plus le virage sera serré. Pour arriver à une grande différence de vitesses, vous pouvez faire tourner

une roue en avant et l'autre en arrière. En fait, pour tourner sur lui-même, il suffit à Thymio de faire tourner ses deux roues à la même vitesse mais dans des sens opposés ! Par exemple, dans ce bloc action moteurs  , le *slider* gauche indique une grande vitesse en arrière, alors que le *slider* droite indique une grande vitesse en avant. Le résultat est que le robot tournera sur lui-même en direction de la gauche, comme indiqué par l'image du robot.

Testez la paire événement-action suivante :



Si vous chargez ensuite ce programme et appuyez sur le bouton central, Thymio tournera sur lui-même. Vous pouvez toujours l'arrêter en appuyant sur  . Vous pouvez maintenant changer les *sliders* et réessayer.

Truc

L'icône du Thymio au centre du bloc action moteurs s'anime dès que vous réglez les *sliders* pour vous donner une idée du mouvement de Thymio ! Lorsque l'animation s'arrête, l'image donne la direction dans laquelle le robot se déplacera quand il exécutera ce bloc action.

Thymio vous aime

Un vrai animal de compagnie ne se contente pas de s'approcher ou de s'éloigner de vous, il vous suit un peu partout ! Pour que Thymio puisse vous suivre le plus fidèlement possible, il faudra ajouter deux paires événement-actions au programme précédant. Si Thymio vous détecte avec son capteur tout à droite, il doit tourner à droite et s'il vous détecte avec son capteur tout à gauche, il doit tourner à gauche.

Programme : **likes.aesl**

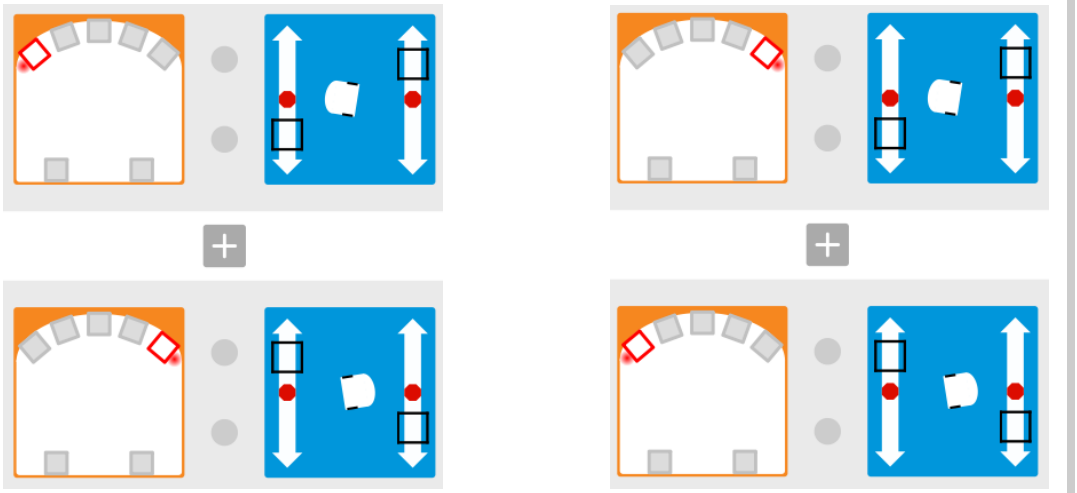
Une façon de faire est illustrée sur la Figure 4.4(a). Vous pouvez essayer différentes vitesses, le faire tourner sur lui-même ou non, afin de trouver le meilleur comportement !

Exercice 4.1

Modifiez le comportement du robot de compagnie pour qu'il bouge en avant quand le programme est exécuté et qu'il s'arrête lorsqu'il détecte le bord de la table (ou une bande de ruban adhésif noir).

Exercice 4.2

Qu'arrive-t-il si vous changez l'ordre des paires événement-actions utilisées dans l'exercice précédent ?



(a) Thymio s'oriente face à vous

(b) Thymio vous évite

FIGURE 4.4 – Programmes pour le robot de compagnie

Thymio ne vous aime pas

Parfois, même le plus fidèle animal de compagnie n'a pas envie de vous suivre. Écrivez un programme qui génère ce comportement.

Programme : **does-not-like.aesl**

Ouvrez le programme du Thymio qui vous aime et inversez l'association des événements avec les actions. Détecter un objet avec le capteur gauche fait tourner le robot à droite, et détecter un objet avec le capteur droite fait tourner le robot à gauche (Figure 4.4(b)).

Exercice 4.3

Les capteurs horizontaux avant sont numérotés 0, 1, 2, 3, 4 de gauche à droite. Les capteurs arrière sont numérotés 5 pour le gauche et 6 pour le droite. À la place d'utiliser les capteurs 0 et 4 comme jusqu'à maintenant :

- Utilisez le capteur 1 pour tourner le robot vers la gauche et le 3 pour tourner vers la droite.
- Utilisez à la fois les capteurs 0 et 1 pour tourner le robot à gauche et à la fois les capteurs 3 et 4 pour tourner le robot à droite.
- Ajouter une paire événement-actions pour les capteurs arrières 5 et 6.

Truc

L'Annexe D montre comment régler les *sliders* à des vitesses précises.

Les capteurs en mode avancé

En mode avancé (Chapitre 7), il existe un quatrième bloc qui permet de spécifier quand les capteurs génèrent des événements, en plus des modes gris, blanc et noir. Référez-vous à l'Annexe D.

Chapitre 5

Thymio est sur une piste

Considérez un entrepôt avec des charriots robotiques qui transportent des objets. Il y a des lignes peintes sur le sol de l'entrepôt et le robot recoit comme instructions de suivre certaines lignes jusqu'à ce qu'il arrive à la zone de stockage de l'objet qu'il transporte. Écrivons un programme qui permette à Thymio de suivre une ligne sur le sol.

Programme : **follow-line.aesl**

Suivre une ligne sur le sol illustre le genre de difficultés rencontrées en robotique : la ligne n'est pas forcément parfaitement droite, de la poussière peut la recouvrir, ou de la saleté peut faire tourner une des roues du robot plus vite que l'autre. Pour suivre une ligne malgré ces difficultés, le robot doit utiliser un *contrôleur*, qui aura pour tâche de définir la puissance à appliquer à chaque moteur en fonction des données reçues par les capteurs.

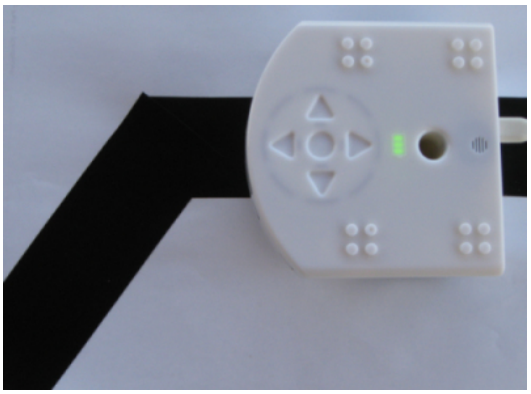
La ligne et le robot

Pour suivre une ligne, nous allons utiliser les capteurs du sol que nous avons déjà utilisé dans le Chapitre 3. Rappelons qu'ils fonctionnent en envoyant de la lumière infrarouge (invisible pour l'oeil humain) et en mesurant la quantité de lumière renvoyée. Si vous poser Thymio sur une couleur claire, beaucoup de lumière sera réfléchie, et donc l'événement sera déclenché. Nous avons donc besoin d'une ligne qui déclenchera un événement quand peu de lumière sera réfléchie  . Ceci est facile à réaliser en imprimant une bande noire, en la peignant ou en utilisant du ruban adhésif noir comme sur la Figure 5.1(a). La ligne doit être assez large pour que les deux capteurs de sol voient du noir quand le robot suit la ligne correctement. Une largeur de 5 centimètres est suffisante pour que le robot suive la ligne même s'il dévie un peu.

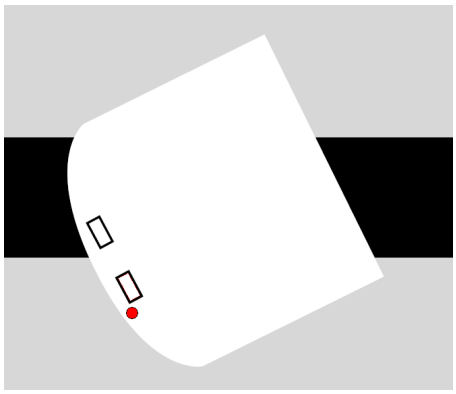
Le premier pas pour faire suivre une ligne à Thymio est de le faire avancer s'il est sur la ligne (les *deux* capteurs de sol détectent du foncé), et de le faire s'arrêter s'il ne se trouve pas sur la ligne (les *deux* capteurs de sol détectent du clair). Voir Figure 5.2(a).

Truc

Assurez-vous d'avoir un câble USB assez long (disons, deux mètres) pour que le robot reste connecté même quand il bouge. Vous trouverez des rallonges dans les magasins d'informatique.

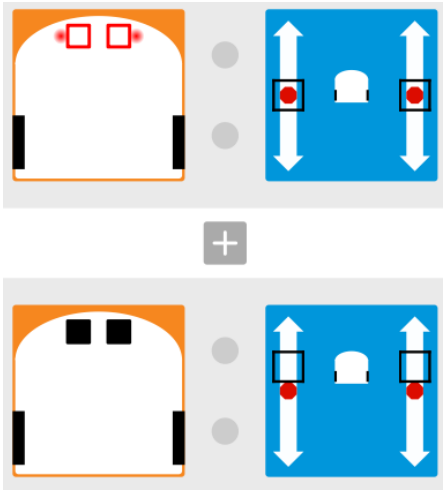


(a) Thymio suit une ligne

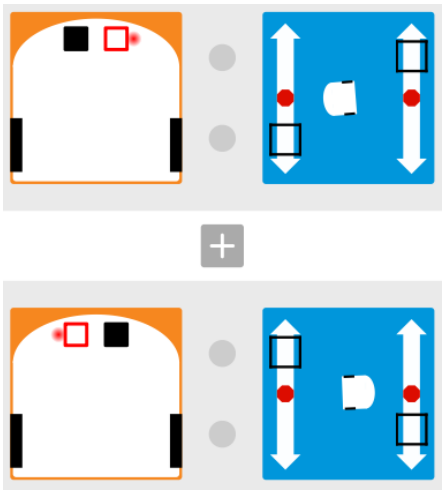


(b) Le capteur gauche est hors de la ligne, le capteur droite est sur la ligne. Le point rouge indique que le capteur gauche mesure beaucoup de lumière réfléchie

FIGURE 5.1 – Thymio sur une ligne de ruban adhésif noir



(a) Thymio s'arrête ou avance



(b) Thymio corrige les déviations

FIGURE 5.2 – Un programme pour suivre une ligne

Votre premier contrôleur

Le prochain pas consiste à programmer le contrôleur qui suit la ligne. Deux paires événement-actions sont nécessaires (Figure 5.2(b)).

— Si le robot sort de la ligne à *gauche* Figure 5.1(b)), le capteur *gauche* détectera le sol alors que le capteur *droite* continuera de détecter la ligne ; dans ce cas le robot doit tourner légèrement à *droite*.

— Si le robot sort de la ligne à *droite*, le capteur *droite* détectera le sol alors que le capteur *gauche* continuera de détecter la ligne ; dans ce cas le robot doit tourner légèrement à *gauche*.

Régler les paramètres

Il est assez facile de comprendre que si Thymio sort de la ligne à gauche, il doit tourner à droite (Figure 5.1(b)). La question est plutôt : de combien doit-il tourner ? Si la rotation est trop douce, le capteur droit pourrait aussi sortir de la ligne avant que le robot ne tourne assez ; si au contraire le tour est trop serré, le robot pourrait sortir de la ligne de l'autre côté. Dans tous les cas, des tours trop forts peuvent être dangereux pour le robot et ce qu'il transporte et qui risque de tomber.

Vous devrez expérimenter avec la vitesse des roues gauche et droite dans chaque bloc action moteurs jusqu'à ce que le robot bouge de manière *fiable*. Par fiable, nous entendons que le robot peut réussir plusieurs fois l'expérience de suivi de ligne. Comme à chaque expérience vous placez le robot sur la ligne à différentes positions et pointant dans differentes directions, il faut tester plusieurs fois pour être convaincu que le programme fonctionne.

Thymio doit-il aller vite lorsqu'il est sur la ligne ou, au contraire, lentement ? En allant vite, vous améliorerez son efficacité pour se déplacer d'un point à un autre mais il risquera de sortir de la ligne avant de pouvoir corriger sa direction. À l'inverse, s'il va trop lentement, personne n'achètera votre robot pour l'utiliser dans en entrepôt.

Exercice 5.1

Thymio s'arrête complètement s'il ne détecte plus du tout la ligne. Modifiez le programme pour qu'il tourne lentement sur lui même vers la gauche jusqu'à ce qu'il retrouve la ligne. Essayez le sur une ligne avec un virage à gauche comme sur la Figure 5.1(a). Essayer d'augmenter la vitesse en avant du robot. Que se passe-t-il lorsque le robot arrive à la fin de la ligne ?

Exercice 5.2

Modifiez le programme de l'exercice précédent pour que le robot tourne à droite quand il arrive au bout de la ligne. Que se passe-t-il ?

Il serait bien de pouvoir se *rappeller* quel capteur était le dernier à perdre le contact avec la ligne afin de faire tourner le robot dans la bonne direction pour retrouver la ligne. Dans le Chapitre 8, nous apprendrons à Thymio à se rappeler ces informations.

Exercice 5.3

Jouez avec différentes formes de parcours :

- Des virages doux
- Des virages serrés
- Des zig-zag
- Des lignes plus larges
- Des lignes plus étroites

Faites la course avec vos amis : Quel robot suit le plus de lignes différentes avec succès ? Pour chaque ligne, quel robot la suit dans le moins de temps ?

Exercice 5.4

Discutez les effets que les modifications suivantes au Thymio auraient sur les capacités du robot à suivre une ligne :

- L'événement de lecture des capteurs de sol devient plus ou moins fréquent.
- Les capteurs sont plus éloignés ou proche l'un de l'autre.
- Il y a plus que deux capteurs de sol sous le robot.

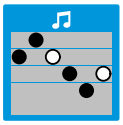
Chapitre 6

Sons et chocs

Jouons un peu avec Thymio. Dans ce chapitre, nous vous montrerons que Thymio peut jouer de la musique et réagir à un son ou à un petite tape amicale !

Thymio mélomane

Thymio possède un synthétiseur de sons qui lui permet de jouer des notes de musique ! Vous pouvez le programmer simplement en utilisant le bloc action :



Programme : **bells.aesl**

Vous ne deviendrez pas le nouveau Beethoven — on ne peut jouer qu'un ensemble de six notes, sur cinq hauteurs et deux longueurs différentes — mais vous pouvez composer une petite musique. La Figure 6.1 illustre deux chansons différentes jouées par Thymio lorsque vous touchez les boutons avant ou arrière. Deux mélodie différente selon l'événement : deux fois la séquence long-court-silence ou alors deux fois la séquence court-long-silence.

Les six petits ronds sont des notes. Un rond noir est une note courte, un rond blanc est une note longue et un espace est un silence. Pour changer la longueur de la note, cliquez sur un rond. Il existe cinq barres horizontales colorées qui représentent les cinq notes. Pour déplacer un cercle vers l'une des barres, cliquez sur la barre au-dessus ou au-dessous du cercle, ou glissez-déposez la note sur l'une des barres.

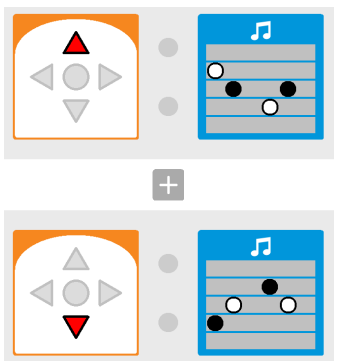


FIGURE 6.1 – Jouer une chanson

Exercice 6.1

Écrivez un programme qui vous permette d'envoyer un code en Morse. Les lettres dans les codes Morse sont encodées par des séquences des sons longs (*traits*) et des sons courts (*points*). Par exemple, la lettre *V* est encodée par trois points suivis par un trait.

Contrôllez votre robot par le son

Thymio a un microphone, il peut donc réagir à un son ! L'événement se déclenche si Thymio entend un son fort, comme par exemple quelqu'un qui tape dans ses mains. La paire événement-actions suivante allumera les lumières en dessous du robot lorsque vous frappez dans vos mains :

Information

Si vous vous trouvez dans un environnement bruyant, vous ne pourrez pas utiliser cet événement, car le niveau sonore sera toujours haut et l'événement s'activera à répétition.

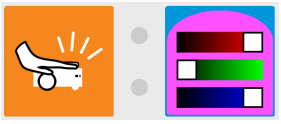
Exercice 6.2

Écrivez un programme qui fasse démarrer le robot quand vous frappez dans vos mains et le fasse s'arrêter lorsque vous touchez un bouton.

Écrivez un programme qui fasse l'opposé : le robot démarre lorsque vous touchez un bouton et s'arrête lorsque vous frappez dans vos mains.

C'est bien Thymio !

Il est important de récompenser votre animal de compagnie quand il est gentil et c'est pareil avec Thymio ! Il peut détecter si vous lui donnez une petite tape sur la tête grâce à l'événement . La paire événement-actions suivante allume les lumières du haut lorsque vous lui donner une petite tape :



Construisez un programme avec cette paire événement-actions et la paire suivante qui allume les lumières du bas du robot lorsque vous tapez dans vos mains :

Programme : **whistles.aesl**

Arrivez-vous à n'allumer que les lumières du haut ? C'est difficile à faire : une tape génère un son qui est assez fort pour faire aussi s'allumer les lumières du bas. Avec un peu de pratique, on arrive à taper le robot assez gentillement pour que le son résultant ne soit pas suffisant pour être considéré.

Exercice 6.3

Écrivez un programme qui fasse avancer le robot jusqu'à ce qu'il touche un mur.

Faites attention à ce que le robot **bouge lentement** afin qu'il ne s'endomage pas.

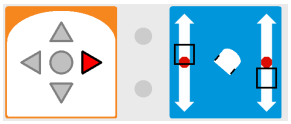
Chapitre 7

Aimer pour un temps (Mode avancé)

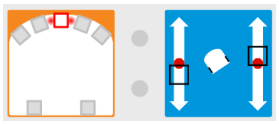
Dans le Chapitre 4, nous avons programmé un robot de compagnie qui nous aimait ou nous fuyait. Considérons un comportement plus avancé : un compagnon timide qui n'arrive pas à se décider s'il nous aime ou pas. Initiallement, le robot tournera en direction de notre main tendue, puis il s'en éloignera en tournant dans l'autre sens. Après un moment, il changera d'avis et reviendra en direction de notre main.

Programme : **shy.aesl**

Lorsque le bouton droite est touché, le robot tourne à droite :



Quand il détecte votre main, il tourne à gauche :



Le comportement de se retourner « après un moment » peut être décomposé en deux paires événement-actions :

— *Quand* le robot commence à se détourner → *démarrer un minuteur* de deux secondes.

— *Quand* le temps du minuteur est écoulé → *tourner* à droite.

Nous avons besoin d'une nouvelle *action* pour la première partie du comportement et d'un nouvel *événement* pour la seconde partie.

L'action qui démarre un *minuteur* est semblable à un réveil. Quand on demande au réveil de sonner à un moment précis, on lui donne une heure absolue, par exemple 7 heures. Mais quand j'utilise mon smartphone comme réveil, il me donne aussi l'heure relative : « alarme dans 11 heures et 23 minutes. » Le minuteur fonctionne ainsi. Vous indiquez au bloc action minuteur un certain nombre de secondes ; quand le nombre de secondes données seront écoulées, le minuteur générera un événement minuteur.

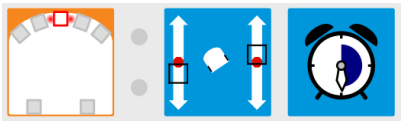
On peut régler le minuteur jusqu'à quatre secondes. Sur le bloc, chaque seconde est représentée par un quart de la montre. Cliquez où vous souhaitez sur la montre ; une petite animation vous indiquera le temps que représente le compte à rebours ; ensuite, la partie correspondante de la montre sera colorée en bleu foncé.

Le mode avancé

Les minuteurs sont disponibles en mode avancé. Cliquez sur  pour entrer en mode avancé.

L'icône deviendra alors  . En cliquant dessus, retournerez en mode débutant.

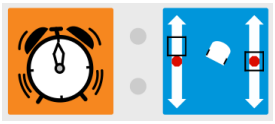
La paire événement-actions pour cette première partie du comportement est :



Lorsque l'événement détectant votre main se produit, il y aura donc deux actions : tourner le robot à gauche et lancer un compte à rebours de deux secondes.

La seconde partie du comportement nécessite un événement se produisant lorsque le compte à rebours arrive à zéro, c'est le bloc événement *temps écoulé* qui montre un réveil sonnant.

La paire événement-actions pour faire tourner à nouveau le robot vers la droite lorsque le minuteur est écoulé est donc :



Exercice 7.1

Écrivez un programme qui fasse avancer le robot à vitesse maximale pour trois seconde lorsque que le bouton avant est touché ; puis qui fasse reculer le robot. Ajoutez une paire événement-actions qui arrête le mouvement en touchant le bouton central.

Information

En robotique, ce genre de compte à rebours s'appelle des *timers*. Ils sont extrêmement utiles dans de nombreuses situations et vous vous en apercevrez très rapidement en créant vos propres comportements pour Thymio.

Chapitre 8

Les états (Mode avancé)

Un programme VPL est composé d'une série de paires événement-actions. *Tous* les événements sont vérifiés périodiquement et les actions appropriées sont effectuées. Ceci limite les programmes que nous pouvons créer ; pour aller plus loin nous avons besoin d'une façon de spécifier que certaines paires événement-actions sont actives à un certain moment, alors que d'autres ne le sont pas.

Par exemple, quand Thymio devait suivre une ligne (Chapitre 5), lorsque le robot sortait de la ligne, nous aurions aimé qu'il tourne à gauche ou à droite afin de rechercher la ligne dans une direction qui dépendait de quel côté il était sorti. Pour cela, deux paires événement-actions sont nécessaires : une pour tourner à gauche lorsque le robot sort à droite de la ligne et une pour tourner à droite lorsqu'il sort à gauche.

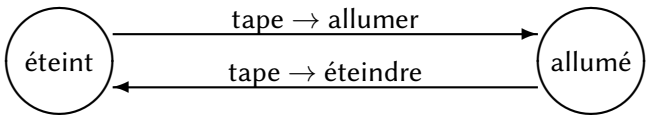
Tape, tape

Dans les programmes que nous avons réalisé jusqu'ici, nous avons souvent *démarré* Thymio en appuyant sur un de ces boutons et *arrêté* Thymio en appuyant sur un autre. Mais regardez votre ordinateur, normalement, il n'a qu'un seul bouton pour l'allumer ou l'éteindre. Le bouton se *rappelle* s'il est dans l'état **allumé** ou l'état **éteint**.

Écrivons un programme qui allume les lumières du robot si vous lui donnez une petite tape et qui les éteigne si vous lui donnez une seconde tape.

Programme **tap-on-off.aesl**

Il est pratique de décrire ce comportement en utilisant un *diagramme d'états* :

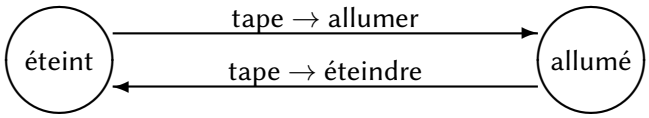


Ce diagramme comprend deux états indiqués par des cercles, **allumé** et **éteint**. Depuis l'état **éteint**, le robot peut aller dans l'état **allumé** et revenir, mais seulement en suivant les instructions sur les flèches. Les instructions décrivent quand une transition d'un état à l'autre peut se produire et comment :

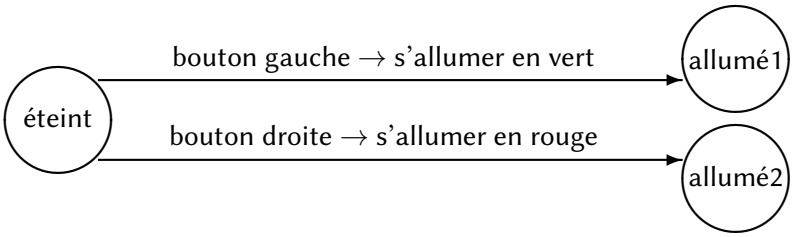
- *Quand* Thymio est dans l'état **éteint *et*** que l'événement *tape* se produit → *allumer* le robot ***et*** aller dans l'etat **allumé**.
- *Quand* Thymio est dans l'état **allumé *et*** que l'événement *tape* se produit → *éteindre* le robot ***et*** aller dans l'etat **éteint**.

L'accent mis sur le mot ***et*** avant la flèche → signifie que deux conditions doivent être remplies pour que la transition se fasse : (a) le robot doit être dans un certain état et (b) l'événement doit se produire. Lorsque les deux conditions sont remplies, alors la transition est prise ce qui fait à la fois changer l'état et exécute l'action écrite après la flèche →.

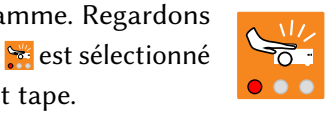
Il est important de réaliser que les deux parties de la condition sont indépendantes. Dans le diagramme ci-dessus (répété ici), l'événement *tape* apparaît deux fois, mais l'action exécutée lorsque cet événement a lieu *dépend de l'état du robot*.



Dans un même état, différents événements peuvent causer différentes actions et des transitions vers de nouveaux états différents. Dans le diagramme suivant, toucher le bouton de gauche dans l'état **éteint** allume la lumière verte du robot et passe l'état du robot à **allumé1**, alors que toucher le bouton droit **dans le même état** entraîne une autre action, allumer la lumière rouge, et un autre changement d'état vers **allumé2**.



Implémenter des diagrammes d'états avec des paires événement-actions

Nous montrons comment *implémenter* le comportement décrit par le diagramme d'état avec des paires événement-actions. Implémenter signifie construire un programme qui fera ce que le diagramme d'états ci-dessus décrit. La Figure 8.1 montre le programme. Regardons maintenant les paires événement-actions une à une. Le cercle gauche du bloc  est sélectionné (et s'affiche en rouge) pour indiquer qu'il s'agit d'un bloc pour l'événement tape.

> **Le bloc tape en mode avancé**
> Le bloc pour l'événement tape est différent en mode avancé car il permet aussi l'utilistation des événements accéléromètre comme décrit au Chapitre 10.

Dans la première paire événement-actions (Figure 8.1(a)), l'événement est composé du bloc événement tape avec une indication d'état  . Un état est indiqué par quatre quartiers d'un cercle, chacun pouvant être soit allumé (orange) ou éteint (blanc). Dans ce programme, nous utiliserons le quartier en haut à gauche pour indiquer si la lumière du haut du robot est éteinte ou allumée. Dans Figure 8.1(a), ce quartier est coloré en blanc et donc la lumière du robot est éteinte. Ainsi, cette paire veut dire : **si** le robot est tapé **et** que le robot est éteint, **alors** allumer la lumière du robot.

(a) Une tape pour allumer la lumière

(b) Une tape pour éteindre la lumière

FIGURE 8.1 – Une tape qui a des résultats différents en fonction de l'état.

Dans la seconde paire événement-actions (Figure 8.1(b)), le quartier est coloré en orange et donc la lumière du robot est allumée. Cette paire veut dire : **si** le robot est tapé **et** qu'il est allumé, **alors** l'éteindre.

Si vous regardez à nouveau le diagramme d'états, vous verrez que seulement la moitié du travail est fait. En effet, en allumant et éteignant le robot, nous devons aussi changer son état d'**éteint** à **allumé** ou d'**allumé** à **éteint**. Ainsi, il nous faut ajouter un bloc action *état*  à chaque paire. Ce bloc change l'état selon si les quartiers sont blancs ou oranges.

Pour résumer, la signification du programme de la figure Figure 8.1 est la suivante :

> *Quand* le robot est tapé *et* que l'état est *éteint*,
> changer l'état à **allumé** *et* **allumer** la lumière du haut.
> *Quand* le robot est tapé *et* que l'état est **allumé**,
> changer l'état à **éteint** *et* **éteindre** la lumière du haut.

Chaque événement entraîne à la fois une action lumière et une action état. Les actions dépendent de l'état dans lequel le robot se trouve, appellé *état actuel*.

Dans combien d'états différents Thymio peut-il être ?

Quand il est utilisé avec un bloc événement état ou action état, chaque quartier peut être :

- **Blanc** : le quartier est *éteint* ;
- **Orange** : le quartier est *allumé* ;
- **Gris** : le quartier n'est pas pris en compte.

Par exemple, dans  , les quartiers en haut à gauche et en bas à droite sont allumés, le quartier en haut à droite est éteint, et le quartier en bas à gauche n'est pas pris en compte. Ceci veut dire que si est associé à un bloc événement, l'événement se produira si l'état est défini soit par :


ou


Comme chacun des quatre quartiers peut être soit allumé soit éteint, il y a $2 \times 2 \times 2 \times 2 = 16$ états :

(éteint, éteint, éteint, éteint)
(éteint, éteint, éteint, allumé)
(éteint, éteint, allumé, éteint)
...
(allumé, allumé, allumé, éteint)
(allumé, allumé, allumé, allumé).

La Figure 8.2(a) montre tous les états possibles.

Information importante

L'état courant du robot est toujours affiché sur le haut du robot par les quatre arcs de cercles lumineux. Par exemple, la Figure 8.2(b) montre le robot dans l'état **(allumé, allumé, allumé, allumé)**.

Information

Lorsqu'un programme démarre, l'état initial est toujours **(éteint, éteint, éteint, éteint)** :

Truc

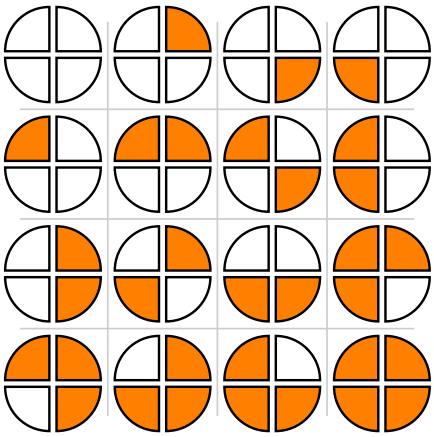
Si vous n'utilisez pas tous les 16 états possibles, mais par exemple que 2 ou 4, vous êtes libre de décider quel quartier vous utiliser pour représenter votre état. Aussi, si par exemple vous avez deux choses différentes à encoder dans l'état, et que chacune d'elle a deux valeurs possibles, vous pouvez utiliser deux quartiers indépendemment. C'est pourquoi la capacité d'*ignorer* un quartier est très utile ! Essayez toujours de rester le plus simple possible.

Attraper la souris

Écrivons un programme qui fasse tourner le robot de droite à gauche à la recherche d'une souris (ou d'un autre objet). Si le robot détecte une souris avec son capteur tout à gauche, il continue la recherche jusqu'à ce que la souris soit détectée avec son capteur tout à droite. Puis, il se positionne en face de la souris, comme sur la Figure 8.3.

Programme : **mouse.aesl**

Le diagramme d'état suivant décrit le comportement du robot :



(a) Tous les états possibles de Thymio

(b) La cercle de lumières indique l'état

FIGURE 8.2 – Les états de Thymio et leur représentation

FIGURE 8.3 – Le robot chat cherche la souris

1. Lorsque le bouton central est touché, le robot entre dans l'état **chercher à gauche** et tourne de droite à gauche.
2. Lorsque le robot est dans l'état **chercher à gauche** et détecte la souris sur son capteur tout à droite, il prend l'état **chercher à droite** et tourne de gauche à droite.
3. Lorsque le robot est dans l'état **chercher à droite** et il détecte la souris avec son capteur central, il prend l'état **trouvé** et s'arrête.

L'essentiel est de remarquer que lorsque le capteur central détecte la souris, le robot s'arrête *seulement si* le robot est dans l'état **chercher à droite**. Sinon (si la souris est détectée par le capteur central en mode **chercher à gauche**), rien ne se produit.

Implémentons ce comportement. L'état du robot sera défini par le quartier supérieur gauche. Choisissons le blanc pour représenter l'état **chercher à gauche** et l'orange pour représenter

l'état **chercher à droite**. Puisque le programme se termine lorsque la souris est détectée dans l'état **chercher à droite**, nous n'avons pas besoin de représenter l'état final **trouvé**. Initialement, tous les quartiers sont **éteints** (blancs).

La paire événement-actions suivante fait tourner le robot à gauche :

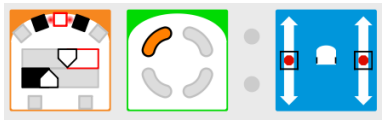
Quand le bouton central est touché, l'état change et devient **chercher à gauche** *et* le robot tourne à gauche.

La prochaine paire événement-actions implémente la deuxième étape :

Quand le robot est dans l'état **chercher à gauche** et que la souris est détectée par le capteur tout à droite, l'état devient **chercher à droite** *et* le robot tourne vers la droite.

Notez que le petit carré à côté de ce capteur est noir pour que l'événement se produise seulement si seul le capteur le plus à droite détecte la souris.

La troisième étape est implémentée dans la paire événement-actions suivante :



Lorsque la souris est détectée par le capteur central dans l'état **chercher à droite**, le robot s'arrête.

Truc

Il vous faudra expérimenter avec la distance de la souris au robot. Si elle est trop proche du robot, les capteurs à côte du capteur central détecteront aussi la souris, alors que l'événement demande qu'ils ne la détecte *pas*.

Exercice 8.1

Écrivez un programme qui fasse danser le robot : il tourne à gauche sur place durant deux secondes, puis tourne à droite sur place durant trois secondes. Ces mouvement se répètent indéfiniment.

Exercice 8.2 (Difficile)

Modifier le programme de suivi de ligne du Chapitre 5 pour que le robot tourne à gauche quand il sort de la ligne par le côté droite, et qu'il tourne à droite quand il sort de la ligne par le côté gauche.

Chapitre 9

Thymio apprend à compter (Mode avancé)

Dans ce chapitre, nous allons montrer comment les états de Thymio peuvent être utilisés pour compter et même pour faire un peu d'arithmétique.

L'implémentation détaillée des projets ne sera pas donné ici. Nous pensons que vous avez désormais assez d'expérience pour les développer vous-mêmes. Les codes source des différents programmes de ce chapitre se trouvent dans l'archive, essayez de ne pas les regarder tout de suite mais plutôt de faire un maximum de chemin de votre côté.

Les projets suivants utilisent l'événement frapper des mains pour changer l'état de Thymio qui est représenté par le cercle de lumières. N'hésitez pas à changer n'importe lequel de ces éléments.

> **Information importante**
> Par défaut, l'état du robot est affiché grâce au cercle de lumière en dessus du robot. La Figure 8.2(b) montre Thymio dans l'état **(allumé, allumé, allumé, allumé)**.

Pair et impair

> **Programme**
> Choisissez l'un des quartiers de l'état. Il sera **éteint** (blanc) si vous tapez dans vos mains un nombre pair de fois et **allumé** (orange) si vous tapez dans vos mains un nombre impair de fois. Presser le bouton central de Thymio ramènera le compte à un nombre pair (puisque zéro est un nombre pair).

Programme : **count-to-two.aesl**

Les termes pair et impair viennent de l'arithmétique *modulo 2*. Dans ce domaine, on compte de 0 à 1 puis de nouveau à 0. Le terme *modulo* est proche de *reste* : si vous tapez dans vos mains 7 fois, et que nous divisons 7 par 2, nous obtenons 3 avec un reste de 1. En *modulo 2*, nous ne gardons que le reste de 1.

Nous appelons aussi ce concept *l'arithmétique cyclique*. Au lieu de compter de 0 à 1, puis de 1 à 2, nous *retournons* à 0 : 0, 1, 0, 1 ...

Ce concept est en fait familier puisqu'il est utilisé lorsque l'on mesure le temps : les secondes et les minutes sont comptées en *modulo 60* et les heures en *modulo 12 ou 24*. La seconde après 59 n'est pas 60 mais 0. Dans la même logique, l'heure après 23 n'est pas 24 mais 0. S'il est 23h00 et que nous nous mettons d'accord pour nous rencontrer 3 heures après, l'heure du rendez-vous est (23+3) modulo 24 = 26 modulo 24 = 2 heures du matin.

Compter en unaire

> Modifiez le programme pour compter en modulo 4. Il y a quatre états possibles : 0, 1, 2, 3. Choisissez trois quartiers, ils représenteront les états 1, 2 et 3 ; l'état 0 sera représenté en mettant tous les quartiers sur éteint.

Cette méthode de représentation de nombre est appelée *unaire* car différents éléments d'un état représentent des nombres différents. Nous utilisons souvent cette représentation pour compter des points par exemple : 𝍸 | représente 6.

Programme : **count-to-four.aesl**

Exercice 9.1

Jusqu'à combien pouvons-nous compter sur le Thymio en utilisant la représentation unaire ?

Compter en binaire

Nous sommes familiarisés avec *les représentations en base*, en particulier avec la base 10 (décimal). Les symboles 256 en base 10 ne représentent pas trois objets indépendants mais bien un seul. Le 6 représente le nombre de 1, le 5 représente le nombre de 10, le 2 représente le nombre de $10 \times 10 = 100$. Si nous additionnons ces nombres nous obtenons deux cent cinquante-six. Grâce à la base 10, nous pouvons représenter facilement de grands nombres. De plus, l'arithmétique que nous avons apprise à l'école nous aide à manipuler de grands chiffres.

Nous utilisons la base 10 parce que nous avons 10 doigts, ce qui rend l'arithmétique en base 10 facil à apprendre. Les ordinateurs n'ont que deux « doigts » (**éteint** et **allumé**), c'est donc cette base qui est utilisée par les ordinateurs. Au début, la base 2 peut paraître étrange mais il est assez facile de s'y faire. Nous utilisons les même symboles qu'en base 10, le 0 et le 1. Au lieu de passer de 1 à 2, nous retournons à 0.

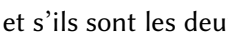
Voici comment un ordinateur compte jusqu'à dix :

$$0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010$$

Prenons le nombre en base 2 suivant : 1101. Nous calculons sa valeur de droite à gauche, comme avec la base 10. Le chiffre le plus à droite représente le nombre de 1, celui d'à côté le nombre de 2, le suivant le nombre de $2 \times 2 = 4$ et le dernier (le plus à gauche) le nombre de $2 \times 2 \times 2 = 8$. Comme nous avons le nombre en base 2 : 1101, nous avons 8 + 4 + 0 + 1, ce qui fait treize.

> **Programme**
> Modifiez le programme pour que Thymio compte en modulo 4 en utilisant la représentation binaire (base 2).

Programme : **count-to-four-binary.aesl**

Nous n'avons besoin que de *deux* arcs du cercle pour représenter les nombres 0, 1, 2, 3 en base 2. Prenons le quartier en haut à droite pour représenter le nombre de 1 — **éteint** (blanc) pour aucun et **allumé** (orange) pour un — et le quartier d'en haut à gauche pour représenter le nombre de 2. Par exemple,  représente le chiffre 1 puisque le quartier supérieur gauche est blanc et le quartier supérieur droit est orange. Si les deux quartiers sont blanc, l'état représente le chiffre 0 et s'ils sont les deux oranges, le chiffre 3. Le nombre 2 est représenté par  , où le quartier supérieur gauche est orange et le quartier supérieur droit est blanc.

Il y a quatre transitions $0 \to 1, 1 \to 2, 2 \to 3, 3 \to 0$, donc quatre paires événements-actions sont nécessaires en addition à une paire événement-actions pour remettre le compteur à zéro lorsque l'on appuie sur le bouton central.

Ignorer les quartiers inutilisés

Les deux quartiers du bas ne sont pas utilisés. Ils sont donc laissés en gris et sont ignorés par le programme.

Exercice 9.2

Développez le programme pour qu'il puisse compter en modulo 8. Le quartier en bas à gauche représentera le nombre de 4.

Exercice 9.3

Jusqu'à combien pouvons-nous compter avec Thymio en utilisant la représentation binaire (base 2) ?

Additionner et soustraire

Écrire un programme pour compter jusqu'à 8 est plutôt long puisqu'il faut créer 8 paires événement-actions, une pour chaque transition de *n* à *n + 1* (modulo 8). Au lieu de procéder de cette façon, nous utilisons des méthodes pour additionner les nombres chiffre à chiffre. En base 10, nous faisons ainsi :

$$\begin{array}{r} 387 \\ +426 \\ \hline 813 \end{array}$$

Et en base 2 :

$$\begin{array}{r} 0011 \\ +1011 \\ \hline 1110 \end{array}$$

En base 2, lorsque nous ajoutons 1 à 1, au lieu de 2, nous obtenons 10. Le 0 est écrit dans la même colonne que le 1 et à côté, à gauche, le nouveau 1 est reporté (c'est la retenue). L'exemple du dessus montre l'addition de 3 (0011) avec 11 (1011), ce qui donne 14 (1110).

Programme
Écrivez un programme qui commence par représenter 0 et qui ajoute 1 à chaque fois que vous tapez dans vos mains. L'addition doit être faite en modulo 16, donc ajouter 1 à 15 retournera à 0.

Programme : **addition.aesl**

Conseils

- En commençant en haut à gauche puis en allant dans le sens contraire des aiguilles d'une montre, les quartiers représenteront le nombre de 1, de 2, de 4 et de 8. Ainsi, le quartier inférieur droit représentera le nombre de 8.
- Si le quartier d'en haut à droite (représentant le nombre de 1) est éteint (blanc) et que Thymio entend un « clap », changez-le simplement à allumé (orange) quel que soit l'état des autres quartiers.
- Si le quartier d'en haut à droite (représentant le nombre de 1) est allumé (orange) et que Thymio entend un « clap », changez-le à éteint (blanc) ensuite, reportez le 1 en retenue. Il y aura trois paires événement-actions, en fonction de la position du *prochain* quartier montrant 0 (blanc).
- Si tous les quartiers sont allumés (orange), la valeur 15 est représentée. Ajouter un à cet état dépasse le modulo et renvoie la valeur à 0. Il vous faut simplement remettre tous les quartiers sur éteint.

Exercice 9.4
Modifiez le programme pour qu'il commence à 15 et soustraie 1 à chaque « clap ». Lorsqu'il arrivera à 0 et qu'il entendra encore un « clap », il devra retourner à 15.

Exercice 9.5
Coller des petites bandes de ruban adhésif noir (ou dessinez simplement) à intervalles réguliers sur une surface blanche. Écrivez ensuite un programme qui fait que Thymio avance et s'arrête lorsqu'il passe sur la quatrième bande.

Cet exercice n'est pas facile. Les bandes de ruban adhésif doivent être assez larges pour que Thymio les détecte mais pas trop pour ne pas qu'il lance plusieurs événements par bande. Il vous faudra faire des essais avec leurs tailles et avec la vitesse de Thymio.

Chapitre 10

Les accéléromètres (Mode avancé)

Tout le monde est familier avec le concept d'*accélération*, par exemple lorsqu'une voiture met les gaz ou freine. Un *accéléromètre* est un outil qui permet de mesurer l'accélération. L'airbag dans les voitures utilise un accéléromètre pour détecter un ralentissement trop rapide à cause d'un accident, ce qui permet de déclancher l'airbag.

Thymio a trois accéléromètres, un pour chacune des directions : avant/arrière, gauche/droite, haut/bas.

Il est difficile d'obtenir des acélérations assez importantes pour être mesurées, sauf dans le cas de la *gravité*, l'accélération qui nous attire vers le centre de la Terre. Dans ce projet, nous allons mesurer l'angle d'inclinaison du Thymio.

Il y a deux événements qui permettent de détecter l'angle du robot par rapport à la Terre :

— : un événement est déclanché lorsque l'angle gauche/droite du robot est à l'intérieur de l'angle blanc du demi-cercle.

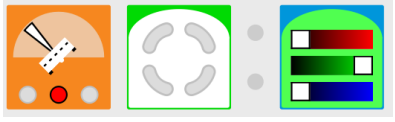
— : un événement est déclanché lorsque l'angle avant/arrière est à l'intérieur de l'angle blanc du demi-cercle.

Au début, l'angle blanc de ces blocs est placé verticalement au-dessus du Thymio, de sorte qu'un événement se produit lorsque le Thymio est à plat, sur le sol ou sur une table. En glissant cet angle avec la souris, on peut choisir d'autres angles ; par exemple, le bloc suivant déclanchera un événement lorsque le robot est penché vers la gauche, plus ou moins à moitié entre la verticale et l'horizontale :

Programme
Tenez le robot face à vous et penchez le vers la gauche et vers la droite. La lumière du dessus du robot doit changer de couleur à chaque intervalle possible de l'angle blanc.

Programme : **measure-angle.aesl**

Construisez un ensemble de paires événement-actions dans lesquels chaque événement est un événement accéléromètre et l'action associée change la couleur du haut du robot :



Créez une liste qui donne les équivalences entre angle et couleur pour que vous puissiez traduire chaque couleur en son angle correspondant.

Les quartiers du bloc événement état sont gris pour que l'événement déclanche l'action quelque soit l'état.

Exercice 10.1

Est-il possible que deux événements utilisent le même angle blanc ?
Combien d'événements avec des angles différents pouvez-vous construire ?

Deuxième partie

Les casse-têtes de Parson

Chapitre 11

Les casse-têtes de Parson

Que sont les casse-têtes de Parson ?

Les *casse-têtes de Parson* sont un type d'exercices qui peuvent aider les étudiants à apprendre à programmer. [1] Un casse-tête de Parson consiste en la donnée d'un objectif pour un programme et d'une série d'instructions dans un langage de programmation. L'exercice consiste alors à placer les instructions dans le bon ordre pour créer un programme qui remplit l'objectif donné. Un casse-tête de Parson peut aussi contenir des *pièges*, comme des instructions incorrectes ou inutiles. L'avantage de ces casse-têtes est que toutes les instructions dont on a besoin sont à disposition des étudiants et ont une syntaxe correcte.

Dans VPL, l'ordre des paires événement-actions ne joue presque aucun rôle. C'est pourquoi les casse-têtes ici seront des programmes avec des paires incomplètes, où le bloc événement, action ou les deux manquent. À droite de chaque paire événement-actions, il y aura deux blocs ou plus ; sélectionnez-en un et tracez une flèche de ce bloc vers la case vide.

Exemple Lorsque le bouton avant est touché, la lumière verte est allumée.

Les casse-têtes

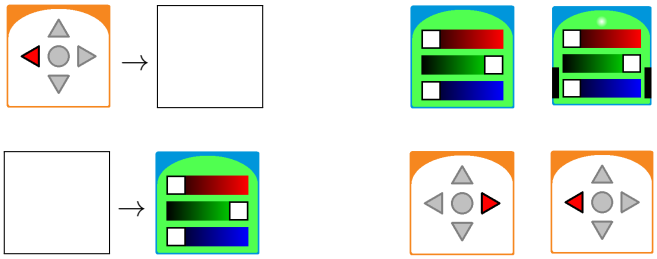
1. Lorsque le bouton droit est touché, la lumière rouge du bas est allumée.

2. Lorsque le bouton droit est touché, la lumière rouge du haut est allumée.

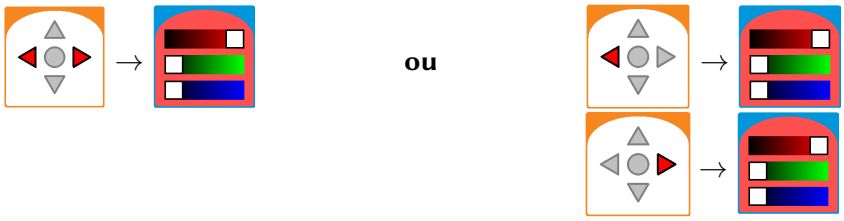
1. Parsons, D. and Haden, P. Parson's programming puzzles : A fun and effective learning tool for first programming courses. *Proceedings of the 8th Australian Conference on Computing Education*, Darlinghurst, Australia, 2006, 157-163.

3. Lorsque le bouton gauche est touché, la lumière verte du bas est allumée.

4. Lorsque le bouton gauche **ou** le bouton droit est touché, la lumière verte du haut est allumée.



5. Lorsque **à la fois** le bouton gauche **et** le bouton droit sont touchés, la lumière rouge du haut est allumée. Choisissez l'un des deux programmes suivants :

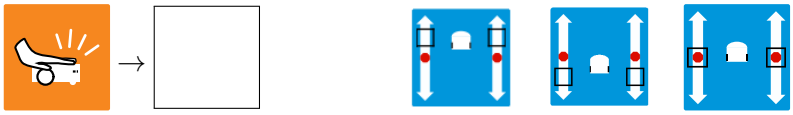


6. Si un objet est détecté par le capteur tout à gauche **seulement**, tournez à gauche.

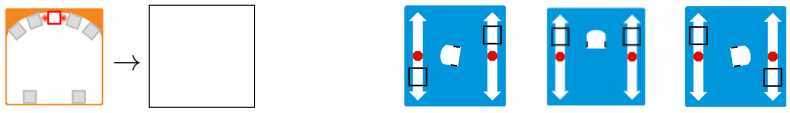
7. Arrêtez le robot lorsqu'il atteint le bord de la table.

8. Lorsque le robot détecte un mur, la lumière rouge du haut est allumée.

9. Lorsque le robot atteint un mur, les moteurs s'arrêtent.

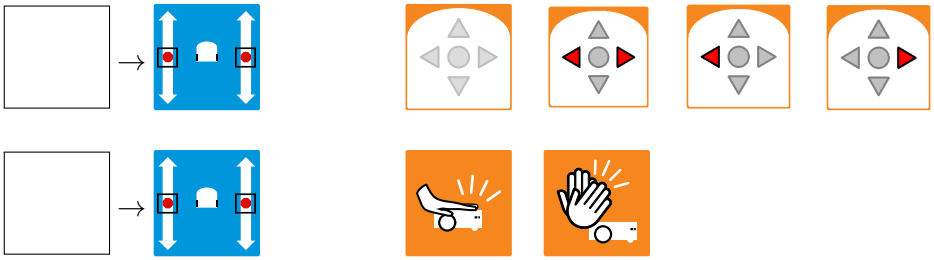


10. Le robot tourne à gauche s'il y a un objet devant le capteur central.

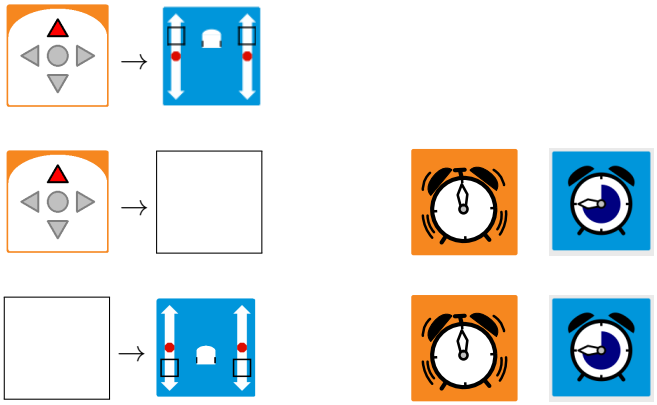


11. Le robot tourne à droite s'il n'y a **pas** d'objet devant le capteur central.

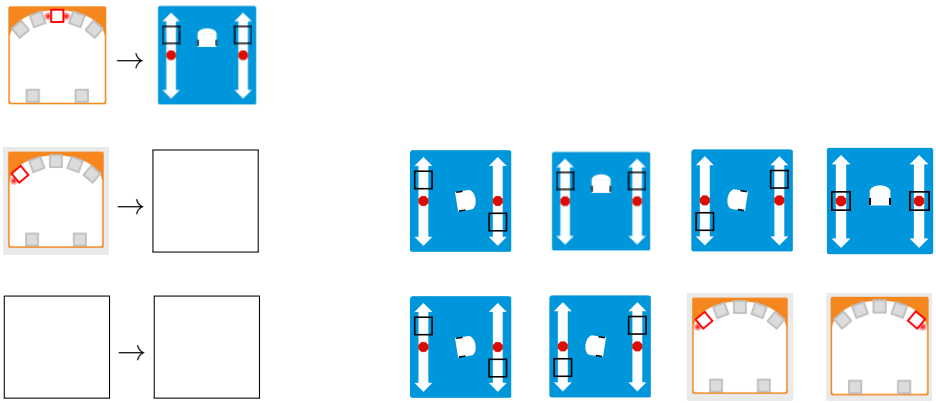
12. Les moteurs sont arrêtés dès que le bouton gauche est touché **ou** si on donne une tape au robot.



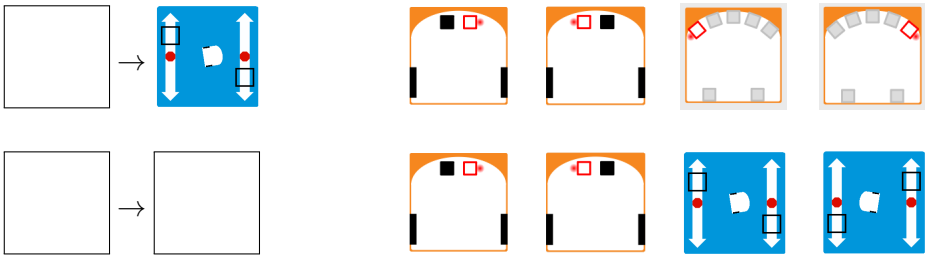
13. Lorsque le bouton avant est touché, le robot avance pendant trois secondes puis recule.



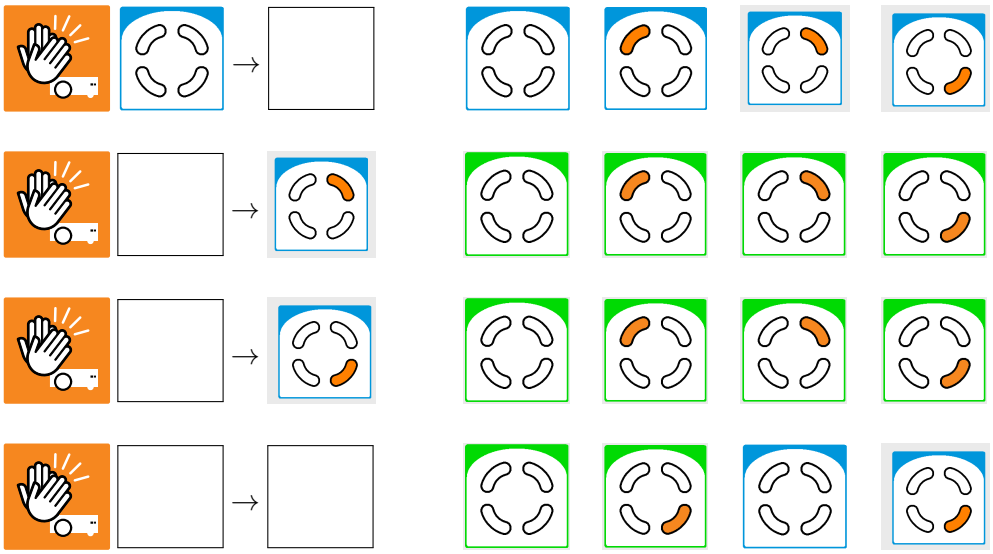
14. Le robot s'approche d'un objet lorsqu'il le détecte par son capteur gauche, droite ou central.



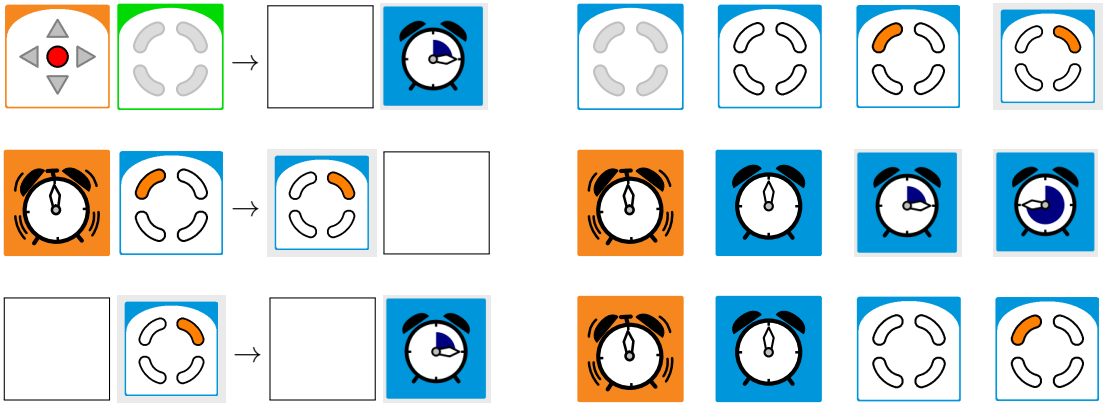
15. Le robot suit une ligne sur le sol. Il tourne à gauche s'il ne détecte plus la ligne sous son capteur droit et tourne à droite s'il ne détecte plus la ligne sous son capteur gauche.



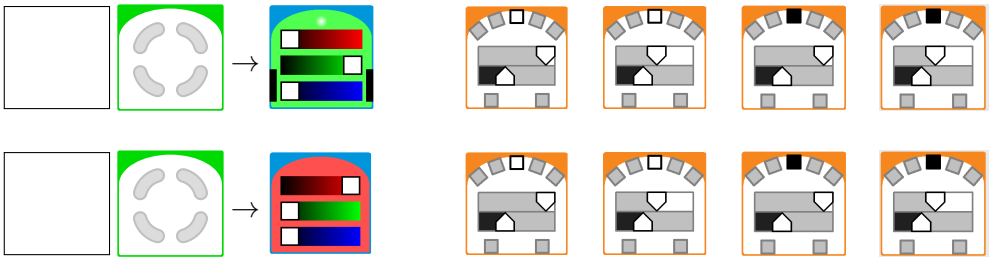
16. Le robot compte dans l'ordre 0,1,2,3,0,1,2,3, . . .chaque fois qu'il détecte un événement frappe les mains.



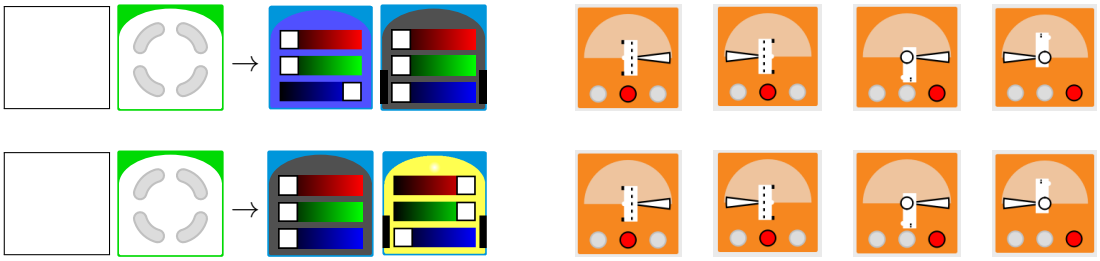
17. Lorsque le bouton central est touché, les arcs de cercles avants droit et gauche s'allument et s'éteignent alternativement chaque seconde.



18. La lumière du bas du robot devient verte lorsqu'il détecte un objet loin de lui et devient rouge lorsqu'il détecte un objet proche de lui.



19. Penchez le robot vers la gauche ; la lumière du haut devient bleue et la lumière du bas s'éteint. Penchez le robot sur son dos ; la lumière du haut s'éteint et la lumière du bas devient jaune.



Troisième partie

Projets

Chapitre 12

Les créatures de Braitenberg

Que sont les créatures de Braitenberg ?

Valentino Braitenberg était un spécialiste en neurosciences qui créa des véhicules virtuels qui présentaient des comportements étonnamment complexes. [1] Ses véhicules ont été largement adoptés dans la robotique éducative. Des chercheurs du MIT Media Lab ont repris ces modèles pour construire ces véhicules en vrai. Ils les ont appelés les *Créatures de Braitenberg*. [2] Ces véhicules étaient construits à partir de *briques programmables* qui étaient les précurseurs des kits robotiques LEGO Mindstorms.

Ce document donne une implémentation de la majorité des créatures de Braitenberg décrites dans rapport le rapport du MIT, adaptées pour le robot Thymio avec la VPL. Les créatures développées au MIT utilisaient des lumières et des capteurs tactiles alors que Thymio utilise principalement les capteurs infrarouges de proximité. Nous avons gardé les noms des créatures qu'on leur avait donné dans le rapport du MIT pour pouvoir établir une équivalence même s'ils ne correspondent pas à l'implémentation pour le Thymio. L'ordre aussi des descriptions des créatures a été conservé bien que celui-ci ne corresponde pas à la difficulté de son implémentation dans VPL.

Dans les descriptions, on utilise, à part mention contraire, l'expression « détecte un objet » pour signifier que le robot détecte un objet par le capteur avant central. Le plus simple pour générer cet événement est de placer sa main devant le capteur central de sorte qu'elle soit détectée par le capteur.

Le code source VPL est disponible dans l'archive. Les fichiers portent les noms en anglais des créatures qu'ils représentent plus l'extension `aesl`. Les noms des fichiers apparaîtront entre parenthèses, à côté du nom du comportement en français. Pour certaines créatures, des comportements supplémentaires sont proposés en exercice. Vous trouverez leur implémentation aussi dans l'archive.

Les créatures

Timide (timid.aesl) Tant que le robot ne détecte aucun objet, il avance. Dès qu'un objet est détecté, il s'arrête.

Indécis (indecisive.aesl) Le robot avance tant qu'il ne détecte aucun objet. Lorsqu'un objet est détecté, il recule. Lorsqu'il se trouve à une certaine distance, le robot *oscille* entre

1. V. Braitenberg. *Vehicles : Experiments in Synthetic Psychology* (MIT Press, 1984).
2. David W. Hogg, Fred Martin, Mitchel Resnick. *Braitenberg Creatures*. MIT Media Laboratory, E&L Memo 13, 1991. http://cosmo.nyu.edu/hogg/lego/braitenberg_vehicles.pdf [en anglais].

marche avant et marche arrière.

Paranoïaque (paranoid.aesl) Lorsque le robot détecte un objet, il avance. S'il ne détecte aucun objet, il tourne à gauche.

Exercice (paranoid1.aesl) Lorsque capteur central du robot détecte un objet, le robot avance. Lorsque c'est le capteur droit (mais pas le capteur central) qui détecte un objet, le robot tourne à droite. Lorsque c'est le capteur gauche (mais pas le capteur central) qui détecte un objet, le robot tourne à gauche.

Exercice (paranoid2.aesl - mode avancé) Tâche similaire à **Paranoïaque**, mais cette fois changer à chaque seconde la direction dans laquelle le robot tourne. **Indice** : Utilisez des états pour retenir la direction actuelle et un minuteur pour changer d'état.

Entêté (dogged.aesl) Lorsqu'un objet est détecté à l'avant, le robot recule et lorsqu'un objet est détecté à l'arrière, le robot avance.

Exercice (dogged1.aesl) Reprenez **Entêté**, mais arrêtez le robot lorsqu'aucun objet n'est détecté.

Désécurisé (insecure.aesl) Si le capteur gauche ne détecte pas d'objet, allumez le moteur droit du robot et éteignez le moteur gauche. Si un objet est détecté par le capteur gauche, allumez le moteur gauche et éteignez le moteur droit. Le robot devrait alors suivre un mur à sa gauche. **Indice** : Voir la référence Annexe B sur les virages avec Thymio.

Déterminé (driven.aesl) Lorsque un objet est détecté par le capteur gauche, il allume le moteur droit et éteint le moteur gauche. Et lorsque un objet est détecté par le capteur droit, il allume le moteur gauche et éteint le moteur droit. Le robot devrait alors s'approcher de l'objet en zigzaguant.

Insistant (persistent.aesl) Le robot avance jusqu'à ce qu'il détecte un objet. Il recule alors pendant une seconde et ensuite avance de nouveau.

Attiré et repoussé (attractive-repulsive.aesl) Lorsqu'un objet approche le robot, il s'enfuit jusqu'à ce qu'il ne le voit plus.

Constant (consistent.aesl - mode avancé) Chaque fois qu'on donne une tappe au robot, celui-ci passe à l'état suivant : d'abord il avance, ensuite il tourne à gauche, puis à droite, puis recule, puis recommemce.

Paniqué (frantic.aesl - mode avancé) La lumière du haut clignote en rouge. **Indice** : Vous pouvez utiliser le bloc événement capteur avec tous les carrés des capteurs en mode gris comme décrit dans Annexe B.

Exercice (frantic1.aesl - mode avancé) Implémentez le clignotement à l'aide du bloc événement boutons au lieu du bloc événement capteurs. Y a-t-il une différence de comportement du robot ? Si oui, quelle en est la cause ?

Observateur (observant.aesl - mode avancé) La lumière du haut est verte quand le capteur droit détecte un objet. La lumière du haut devient rouge lorsque le capteur gauche détecte un objet. Une fois la lumière du haut allumée, le robot attend trois secondes avant de s'éteindre ; pendant ce temps, la lumière ne change pas.

Chapitre 13

Le lièvre et le renard

Ce chapitre donne les instructions pour un gros projet (mon programme contient 7 paires événement-actions, chacune d'elles avec 2–3 actions). Vous devriez entre temps avoir acquis assez d'expérience dans VPL pour réaliser ce projet vous-même. Nous vous donnerons toutes les caractéristiques du comportement à développer sous la forme d'une liste d'instructions. Nous vous suggérons de construire le programme en implémentant chaque contrainte une à une.

Le problème [1] Le robot est un lièvre qui marche dans la forêt. Un renard le chasse et essaie de l'attraper depuis derrière. Le lièvre remarque le renard, se retourne et attrape le renard.

Instructions

Pour chaque événement, nous donnons la couleur qui doit apparaître sur le haut du robot lorsque l'événement a lieu.

1. Le bouton avant est touché : le robot avance (bleu).
2. Le bouton arrière est touché : le robot s'arrête (éteint).
3. Si le robot détecte le bord d'une table, il s'arrête (éteint).
4. Si le capteur arrière gauche détecte un objet, le robot tourne rapidement vers la gauche (dans le sens contraire des aiguilles d'une montre) jusqu'à ce que l'objet soit détecté par le capteur avant central (rouge).
5. Si le capteur arrière droit détecte un objet, le robot tourne rapidement vers la droite (dans le sens des aiguilles d'une montre) jusqu'à ce que l'objet soit détecté par le capteur avant central (vert).
6. Lorsque l'objet est détecté par le capteur avant central, le robot avance rapidement pendant une seconde (jaune) et s'arrête (éteint).

Programme : **rabbit-fox.aesl**

1. Cette histoire est inspirée d'une blague (en anglais) bien connue des doctorants

Chapitre 14

Lire des codes-barres

Les codes-barres sont largement utilisés dans les supermarchés et ailleurs pour identifier des objets. L'identifiant est un nombre ou une suite de symboles unique à chaque type d'objet. Cet identifiant est utilisé pour retrouver dans la base de données des informations sur cet objet comme le prix par exemple. Construisons maintenant un lecteur de code-barres grâce au Thymio.

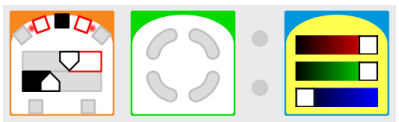
Instructions

1. Mesurez soigneusement la distance entre deux capteurs frontaux ainsi que la largeur d'un de ces capteurs. Utilisez une feuille de carton pliable, du ruban adhésif noir et des bandes de papier d'aluminium pour créer un bricollage qui contient différentes combinaisons de bandes noires ou blanches.

2. Chacune des configurations pour les trois capteurs horizontaux centraux représentent un code différent. (Combien de codes différents peut-on créer ?) Implémentez des paires événement-actions pour certains voire tous ces différents codes et créez différentes couleurs sur le haut du Thymio selon quel code est identifié.

Conseils :

Seuls les trois capteurs centraux seront utilisés ; mettez les carrés des capteurs horizontaux en mode gris. Les carrés des capteurs centraux doivent être blancs quand l'on souhaite détecter une bande d'aluminium blanche et ils doivent être noirs quand l'on souhaite détecter le fond noir du ruban adhésif. Ainsi, la paire événement-actions suivante affiche du jaune pour le code `allumé-éteint-allumé` :



La solution dans l'archive identifie tous les codes avec deux bandes d'aluminium ainsi que le code sans aucune barre d'aluminium.

Programme : **barcode.aesl**

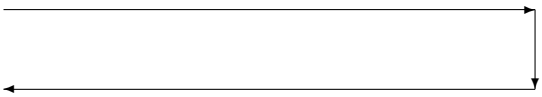
Chapitre 15

Nettoyer le sol

Marre de nettoyer votre maison ? Il existe maintenant les *robots aspirateurs* qui font le travail à votre place ! Le robot traverse systématiquement votre appartement pour aspirer les saletés tout en évitant les obstacles.

Instructions

Lorsque l'on appuie sur le bouton avant, le robot traverse la chambre d'un bout à l'autre, tourne, avance un petit bout puis retourne vers le côté original :



Conseils

Le robot doit implémenter trois tâches distinctes : (1) traverser la chambre (vers la droite ou vers la gauche), (2) tourner à droite, (3) avancer un peu. Ces tâches sont exécutées dans l'ordre suivant :

Il faudra utiliser des états différents pour mémoriser quelle tâche est en train d'être exécutée. Dans chaque tâche, la direction et la quantité de mouvement sont déterminées par la vitesse des moteurs droite et gauche et par la durée pendant laquelle ils sont allumés. C'est pourquoi chaque tâche sera implémentée grâce à une paire événement-actions où l'événement sera la fin du minuteur de la tâche précédente et les actions contrôleront le changement d'état. Il faudra pour cela : (1) changer l'état ; (2) changer les vitesses des moteurs gauche et droite ; (3) lancer un nouveau minuteur. Pour lancer le programme, on utilisera un événement bouton.

Il vous faudra ajuster les vitesses et la longueur des minuteurs pour que le robot suive la trajectoire rectangulaire désirée.

Programme : **sweep.aesl**

Vous pouvez encore vous amuser à ajouter des couleurs différentes pour les différents états : vert quand le Thymio va tout droit, jaune quand il tourne et rouge quand il est arrêté.

Programme : **sweep1.aesl**

Chapitre 16

Mesurer sa vitesse

Instructions

Mesurez la vitesse du Thymio dans différentes configurations des moteurs des roues. Placez une bande de ruban adhésif noire sur une surface claire comme lorsque le Thymio suivait la ligne (Chapitre 5). Placez le robot à l'un des bouts de la bande. Implémentez le comportement suivant :

— Le robot commence à avancer lorsque l'on appuie sur le bouton central.

— Lorsque le robot détecte le début de la bande avec ces capteurs du bas, lancez un minuteur d'une seconde.

— Quand le minuteur s'est écoulé, changez la couleur du haut et relancez le minuteur d'une seconde.

— Quand le Thymio arrive au bout de la bande, éteignez les moteurs.

Lancez le programme et comptez combien de fois la couleur change. Ce sont le nombre de secondes qu'il a fallu au robot pour traverser la bande. Divisez la longueur de la bande par le nombre de secondes obtenu pour obtenir la vitesse. Si par exemple la bande mesure 30 centimètres et la couleur change 6 fois, alors la vitesse du robot est 30/6=5 centimètres par seconde.

Essayez de modifier la configuration des moteurs et la longueur de la bande.

Conseils

Faites une liste de couleurs, par exemple 1=rouge, 2=bleu, 3=vert, 4=jaune, etc et utilisez cette liste pour déduire le nombre de secondes à partir de la couleur du robot.

Utilisez des états pour mémoriser la couleur actuelle et les prochaines couleurs. Par exemple, dans l'état 3, la couleur est vert ; quand le minuteur est écoulé *et* l'état est 3, passez à l'état 4, affichez la couleur jaune et relancez le minuteur. Pour chaque événement minuteur, il y a trois actions.

Programme **measure-speed.aesl**

Chapitre 17

Attrapez les chauffards

Instructions

Aidez la police à arrêter les conducteurs qui roulent trop vite. Calculez la vitesse en mesurant la vitesse que la voiture parcourt en un temps donné.

Le robot détecte les objets qui se déplacent de sa gauche vers sa droite grâce à ses capteurs avants. Allumez une couleur différente selon la distance qu'a parcourue un objet pendant la seconde qui a suivi la détection de l'objet par le capteur tout à gauche.

Conseils

- Dans l'état initial, la détection d'un objet par le capteur avant tout à gauche lance un minuteur d'une seconde.
- Quand le minuteur a expiré, passez à un nouvel état que l'on va appeler l'état *mesurer*.
- Construisez quatre paires événement-actions, une pour chacun des quatre autres capteurs avants ; ces événements n'auront lieu que lorsque le robot sera dans le mode *mesurer*. Lorsque l'un des capteurs détecte un objet, il affiche en haut une couleur associée à ce capteur.
- Assurez-vous qu'une paire événement-actions n'a lieu que lorsqu'un capteur détecte un objet et qu'aucun autre capteur ne détecte d'objet.

Programme : **speeders.aesl**

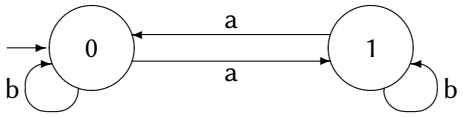
Chapitre 18

Automates finis

Un *automate fini* (*finite automaton* (FA) en anglais) est une machine abstraite capable d'exécuter des calculs. Ces automates sont extrêmement importants dans de nombreux domaines de l'informatique. [1] Considérons une chaîne finie de caractères a ou b :

aabbbababbaba

L'objectif de ce problème est de lire une chaîne de caractères comme celle-ci, puis de déterminer si le nombre de a est pair ou impair. Un FA pour résoudre ce problème contient deux états : l'état 0 si le nombre de a lus jusqu'ici est pair, l'état 1 dans le cas contraire. Ce FA est représenté dans le diagramme suivant. Il contient deux états, 0 et 1, et des transitions d'un état à l'autre, notées a et b.



Lorsque notre FA dans un certain état lit un caractère de la chaîne, il change d'état suivant les transistions du schéma. Si le nombre de a lus jusqu'ici est pair (état 0) et que le FA lit un a, alors l'état devient 1. Et inversément, si le nombre de a lus jusqu'ici est impair (état 1) et que le FA lit un a, le FA passe à l'état 0. Si un b est lu par contre, l'état ne change pas puisque le nombre de a n'a pas changé.

Le FA démarre dans l'état 0 puisqu'au début zéro a ont été lus, ce qui est un nombre pair. L'état initial est indiqué sur le schéma par une petite flèche.

Instructions [2]

Imprimez le fichier `fa-path-alternate.pdf`. Il contient l'image suivante : [3]

1. Une définition formelle des automates finis peut se trouver par exemple dans J.E. Hopcroft, R. Motwani, J.D. Ullman. *Introduction to Automata Theory, Languages and Computation*, Pearson, 2013.
2. Ces instructions ainsi que la solution ont été inspirés par le projet de *Light Painting*.
3. Les fichiers tracé se trouvent dans le dossier **images** de cette archive.

La ligne dégradée permet au robot d'avancer droit et de ne pas sortir du tracé à gauche ou à droite. La chaîne de caractères est représentée ici par des carrés noirs pour les a et blancs pour les b. Cette image représente la chaîne de caractères $bababab$.

Placez le robot à gauche de l'image, avant le premier carré. Il doit être tourné vers la droite et le capteur du sol droit doit être au milieu de la ligne dégradée. Voici le comportement du robot :

1. Appuyer sur le bouton avant démarre le robot. La lumière du haut doit être éteinte, l'état est initialisé (voir ci-dessous) et un minuteur est démarré.
2. Appuyer sur le bouton central arrête le robot.
3. Le robot tourne vers la droite. Il utilise le capteur du sol *droit* pour détecter si il commence à dévier vers la droite ou vers la gauche. Si le robot tourne vers la droite, le capteur détectera un niveau de lumière plus bas (plus noir) ; le robot tourne alors vers la gauche. Si le robot tourne vers la gauche, le capteur détectera plus de lumière (plus blanc) ; le robot tourne alors vers la droite.
4. Lorsque le minuteur est écoulé, la valeur du capteur du sol *gauche* est relevée. Si le robot est alors sur un carré blanc (il lit un b), la lumière du haut s'allume en rouge et le minuteur est relancé. Si le robot est alors sur un carré noir (il lit un a), la lumière du haut s'allume en vert, le minuteur est relancé et l'état change de pair à impair, ou d'impair à pair.

Conseils

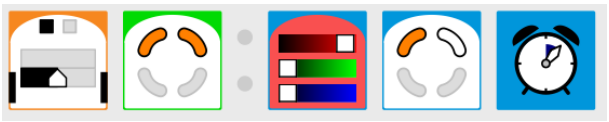
Le Thymio utilisera trois des quatre quartiers dans les événements et actions d'état :

— Le quartier supérieur gauche indique si le programme a démarré ou non.

— Le quartier supérieur droit indique si oui ou non il faut relever la couleur (noir ou blanc) du carré sur lequel se trouve le robot.

— Le quartier inférieur gauche retient si le nombre de a lus est pair ou impair.

Voici un exemple d'une paire événement-actions.



si le capteur gauche détecte peu de lumière (un carré noir) *et*
    le programme a déjà démarré *et* la couleur du carré doit être relevée *et*
    jusqu'ici le nombre de a lus est pair, alors
        allumer le haut en rouge
        changer d'état : le nombre de a lus est impair *et*
            ne plus relever la couleur du carré
        relancer le minuteur

Programme : **fa.aesl**

Vous devrez essayer différentes vitesses de moteur et longueurs du minuteur pour pouvoir détecter fiablement les carrés noirs et blancs.

Exercice Imprimez le fichier `fa-path-blank.pdf` qui contient une image avec des carrés tous blancs. Utilisez un marqueur noir pour colorier certains carrés et essayer une nouvelle chaîne de caractères. Verifiez en particulier que votre programme fonctionne lorsque plusieurs carrés de suite sont noirs, représentant une chaîne de caractères avec plusieurs a à la suite.

Exercice Modifiez votre programme pour qu'il détermine le reste (0, 1 ou 2) du nombre de a divisé par 3.

Créer ses propres tracés

> **Attention !**
> Cette section montre comment modifier le tracé (la ligne dégradée et les carrés), mais pour cela il faut que vous maîtrisiez le langage de mise en page de documents LaTeX.

Les fichiers `fa-path-*.tex` dans l'archive contiennent le code source LaTeX.

Les instructions suivantes créent une ligne dégradée de 2 cm de largeur et de 23 cm de longueur : [4]

```
\shade[left color=black,right color=white] (0,0) rectangle +(2,23);
```

On peut dessiner les carrés noirs et blancs avec les instructions suivantes :

```
\foreach \a in {1, 3, 5, 7}
  \filldraw[color=black] (\offset,\height*\a) rectangle +(\width,\height);

\foreach \a in {0, 2, 4, 6, 8}
  \draw (\offset,\height*\a) rectangle +(\width,\height);
```

Vous pouvez changer la liste de nombres dans les instructions `foreach` pour indiquer quels carrés doivent être blancs et lesquels doivent être noirs.

Les instructions `filldraw` et `draw` utilisent pour les dimensions des paramètres qui peuvent être modifiés facilement :

```
\setlength{\height}{2.4cm}  % Hauteur d'un carré
\setlength{\width}{3cm}     % Largeur d'un carré
\setlength{\offset}{2.3cm}  % Séparation entre carrés et ligne dégradée
```

Compilez le fichier avec `pdflatex` et imprimez le en utilisant `Adobe Reader` ou `SumatraPDF`. L'image est créée en format portrait mais vous pouvez fixer la feuille sur la table dans les deux sens. On peut accrocher plusieurs pages ensemble sur une table pour représenter des chaînes de caractères plus longues.

4. Ces instructions utilisent la bibliothèque graphique Ti*k*Z.

Chapitre 19

Des capteurs avec plusieurs seuils

Le mode avancé permet de définir les événements capteurs de trois manières différentes, comme décrit dans l'Annexe D : l'événement est déclanché lorsque la lumière réflechie est en-dessous d'un certain seuil (noir), l'événement est déclanché lorsque la lumière réflechie est au-dessus d'un certain seuil (blanc), et l'événement est déclanché lorsque la lumière réflechie est entre deux seuils (gris foncé) :

Instructions

Créez un programme dans lequel le robot s'approche d'un objet, d'abord à grande vitesse, puis il ralentit au fur et à mesure qu'il s'approche de l'objet jusqu'à s'arrêter complètement lorsqu'il en est très proche.

Conseils

- Utilisez trois paires événement-actions, une pour chacun des types d'événement capteurs.
- Glissez soigneusement les *sliders* (voir Annexe D) de sorte à ce que le seuil supérieur d'un intervalle corresponde au seuil inférieur du prochain intervalle.
- Ajoutez un bloc couleur à chacune des paires pour pouvoir remarquer les changements de vitesse.
- Utilisez du ruban adhésif réfléchissant pour augmenter la portée des capteurs, comme expliqué dans l'Annexe B.

Programme : **slow.aesl**

Chapitre 20

Plusieurs Thymio

Vous pouvez contôler deux ou plus de Thymio simultanément.

Instructions

Placez deux robots T1 et T2, un en face de l'autre. T1 poursuit T2 ; lorsque T1 détecte qu'il est proche de T2, il s'arrête. Si T2 détecte que T1 est proche de lui, T2 recule jusqu'à ce qu'il ne voit plus T1.

Conseils

- Les programmes de T1 et T2 ont deux paires événements-action : l'événement de la première est la détection d'un objet par le capteur central horizontal, l'événement de la seconde est l'absence de détection d'un objet. Mais les programmes de T1 et T2 diffèrent par les actions associées aux événements.
- Connectez les deux Thymios T1 et T2 à l'ordinateur et allumez-les. Lancez deux fois VPL. Dans la fenêtre qui permet de choisir le robot qu'utilisera VPL (Figure 1.2), vous devriez voir à la fois T1 et T2 ; sélectionnez T1 dans la première instance de VPL et T2 dans la seconde. Ouvrez et lancez le programme `chase` pour T1 et le programme `retreat` pour T2.

Expériences

- Que se passe-t-il si vous échangez les programmes : lancez `retreat` pour T1 et `chase` pour T2 ? Expliquez.
- En mode avancé, testez différents paramètres des seuils des capteurs.

Programme : **chase.aesl**, **retreat.aesl**

> ⭐ **Communiquer entre les robots**
> Plusieurs robots Thymio peuvent s'envoyer des messages. Cette fonctionnalité est disponible dans le langage AESL et l'environnement Studio.

Quatrième partie

De la programmation visuelle à la programmation textuelle

Chapitre 21

Apprendre le langage AESL à partir des programmes VPL

Félicitations ! Vous êtes expert(e) de la programmation visuelle (VPL). Vous pouvez maintenant vous lancer dans la programmation textuelle avec l'*environnement de programmation Studio* et son langage de programmation, l'*Aseba Event Scripting Language (AESL).*

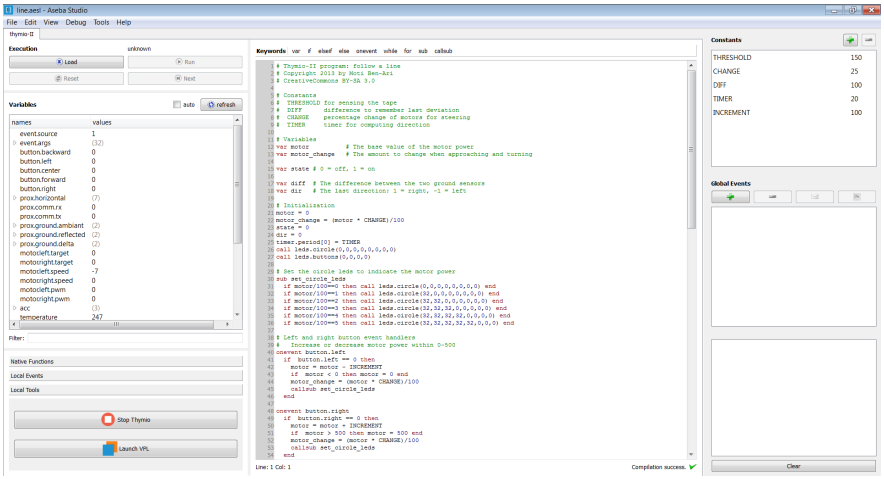


FIGURE 21.1 – L'environnement d'Aseba Studio

VPL traduit les programmes graphiques (les paires événement-actions) en programmes textuels AESL qui apparaîssent d'ailleurs sur le côté droit de la fenêtre VPL (numéro 6 dans la Figure 1.3 de la page 9). Ce tutoriel utilise des programmes VPL des chapitres précédents et décrit leur programme AESL correspondant. Vous utiliserez vos connaissances de ces programmes VPL pour apprendre les principes fondamentaux de la programmation AESL.

La programmation utilisant Aseba Studio est aussi basée sur les concepts d'événements et d'actions. Comme les programmes VPL sont traduits en programmes AESL, tout ce que vous avez appris dans ce tutoriel est aussi disponible dans Studio, mais vous avez maintenant la flexibilité d'un langage de programmation complet avec des variables, des expressions et des structures de contrôle.

Lorsque vous travaillez dans Aseba Studio, vous pouvez ouvrir VPL en cliquant sur le bouton **Lancer VPL** dans l'onglet *Outils* en bas à gauche de la fenêtre. Vous pouvez importer des programmes VPL vers Aseba Studio simplement en ouvrant le fichier du programme.

Les sections avec une * présentent des concepts de la programmation AESL qui dépassent les possibilités de VPL. Elles peuvent être sautées si vous lisez ce tutoriel pour la première fois.

Documentation

Pour en savoir plus sur Aseba Studio et AESL, rendez-vous sur la page *Programmer avec du texte* disponible sur https://www.thymio.org/fr:asebausermanual. Vous y trouverez la documentation concernant :

— L'environnement de programmation Studio
— Le langage de programmation AESL
— L'interface pour le robot Thymio. (il existe une feuille de référence pour l'interface).
— La bibliothèque contenant toutes les fonctions natives présentes dans AESL.

Il y a aussi une archive qui propose différents projets intéressants à réaliser avec AESL avec le code source de la solution proposée.

L'interface du Thymio

Voici le programme `whistles.aesl` du Chapitre 6 avec une partie du programme AESL correspondant :

```
onevent tap
  call leds.top(32,0,32)

onevent mic
  call leds.bottom.left(0,32,32)
  call leds.bottom.right(0,32,32)
```

Traitement des événements

Lorsqu'un événement tape est déclenché, la lumière du haut est allumée en une couleur appelée *magenta*, et lorsqu'un événement frapper des mains est déclenché, la lumière du bas est allumée en une couleur appelée *cyan*. Dans AESL, ce qui correspond aux paires événement-actions de VPL est un système de traitement des événements qui est introduit par le mot-clé `onevent` (en anglais, *on event* signifie « lors de l'événement ») Vous trouverez une liste des événements dans la table au bas de la documentation pour l'interface de programmation de Thymio.

Les lignes qui suivent `onevent` forment le corps principal du traitement de l'événement `tap`, respectivement `mic`. Ils correspondent aux blocs action à droite des blocs événement dans VPL.

Lorsqu'un événement tape est déclenché, la *fonction d'interface* `leds.top` est *appelée*. La fonction prend trois *paramètres* qui spécifient l'intensité de rouge, de vert et de bleu de la LED. Les valeurs de celles-ci varient entre 0 (éteinte) et 32 (complètement allumée). Combiner du rouge et du bleu permet d'obtenir du magenta.

L'événement VPL frapper dans les mains correspond à l'événement `mic` (une forme abrégée pour microphone). Lorsque l'événement est déclenché, les LEDs du bas sont allumées. Dans VPL, un bloc action allume les deux LEDs de la même couleur alors que dans AESL, les LEDs droite et gauche peuvent être allumées indépendamment l'une de l'autre. Dans notre cas, nous allumons les deux LEDs avec les composantes vertes et bleues à pleine intensité, ce qui donne du cyan.

Affecter une valeur à une variable

Regardez à nouveau le programme AESL dans la fenêtre VPL. Les deux premières lignes sont :

```
# setup threshold for detecting claps
mic.threshold = 250
```

Une ligne qui commence avec un **#** est appelée un *commentaire*. Les commentaires n'ont aucune influence sur le programme ; on les utilise pour donner des informations au lecteur du programme. Dans ce cas, le commentaire explique que l'événement frapper les mains est déclenché lorsque l'intensité du son est plus grande qu'un certain *seuil*. La deuxième ligne du programme spécifie que l'événement est déclenché lorsque l'intensité du son (qui est entre 0 et 255) est plus grand que 250.

Dans VPL, ce seuil est intégré à l'événement et il ne peut pas être changé, mais dans un programme textuel, vous pouvez le changer en *affectant* une valeur à une variable :

```
mic.threshold = 180
```

Ceci signifie que la *valeur* à droite du **=** est copiée vers la *variable* à gauche. La variable `mic.threshold` est prédéfinie pour le robot Thymio.

Initialisation du Thymio

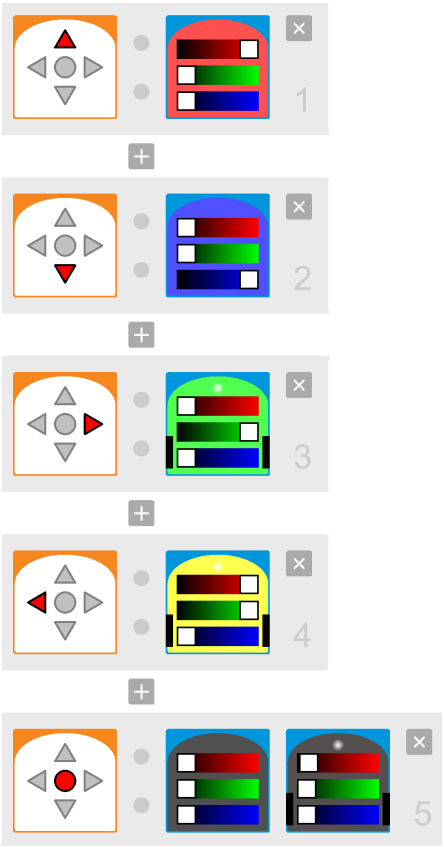
Au début de chaque programme, VPL insère automatiquement une suite d'instructions pour éteindre toutes les LEDs et le son.

```
# reset outputs
call sound.system(-1)
call leds.top(0,0,0)
call leds.bottom.left(0,0,0)
call leds.bottom.right(0,0,0)
call leds.circle(0,0,0,0,0,0,0,0)
```

Cette *initialisation* n'est pas visible dans le programme VPL. Dans un programme textuel, nous recommandons d'inclure ces instructions, bien qu'elles ne soient pas nécessaires.

Plusieurs alternatives

Le programme `colors-multiple.aesl` du Chapitre 2 change la couleur des LEDs du haut et du bas lorsque les boutons sont touchés :




```
onevent buttons
  when button.forward == 1 do
    call leds.top(32,0,0)
  end
  when button.backward == 1 do
    call leds.top(0,0,32)
  end
  when button.right == 1 do
    call leds.bottom.left(0,32,0)
    call leds.bottom.right(0,32,0)
  end
  when button.left == 1 do
    call leds.bottom.left(32,32,0)
    call leds.bottom.right(32,32,0)
  end
  when button.center == 1 do
    call leds.top(0,0,0)
    call leds.bottom.left(7,0,0)
    call leds.bottom.right(7,0,0)
  end
```

Dans le programme AESL, *un seul* événement est déclenché chaque fois qu'un des cinq boutons est touché. L'action du traitement d'événement introduit par `onevent buttons` dépend du bouton touché. Nous allons donc nous référer à la valeur des *variables bouton* pour sélectionner l'action à exécuter. Les instructions :

```
when button.forward == 1 do
  call leds.top(32,0,0)
end
```

signifient : *quand* (« when » signifie « quand » en anglais) la valeur de la variable `button.forward` (*button.forward* signifie « bouton.avant» en anglais) est un, *alors* exécute les actions écrites entre les mots-clés `do` (qui signifie « exécute » en anglais) et `end` (qui signifie « fin » en anglais). Il y a cinq variables `bouton`, une pour chacun des boutons. La valeur d'une variable bouton vaut 1 si le bouton est touché et 0 si le bouton est relâché. Dans ce programme, il y a cinq structures introduites par `when`, une pour chaque bouton. Une ou deux actions sont exécutées si l'expression dans une structure `when` *devient* vraie.

> **La programmation textuelle et l'anglais**
>
> La grande majorité – si ce n'est la totalité – des langages de programmation utilisent des mots-clés en anglais, la langue de référence pour les codeurs du monde entier. Ainsi, nous avons déjà vu les mots-clés `onevent`, `when`, `do` et `end` qui nous viennent tous de l'anglais. Nous en verrons encore quelques autres. Si vous ne comprenez pas l'anglais, il vous faudra mémoriser ces quelques mots, mais, rassurez-vous, ils ne sont pas nombreux.

Un ou plusieurs événements*

L'interface Thymio inclut aussi des événements distincts pour chaque bouton, en plus de l'événement `buttons` qui est déclenché chaque fois qu'un bouton est pressé ou relâché. Nous pourrions aussi implémenter ce programme avec de multiples événements et sans structures `when` ni variables bouton :

```
onevent button.forward
  call leds.top(32,0,0)

onevent button.backward
  call leds.top(0,0,32)

onevent button.right
  call leds.bottom.left(0,32,0)
  call leds.bottom.right(0,32,0)

onevent button.left
  call leds.bottom.left(32,32,0)
  call leds.bottom.right(32,32,0)

onevent button.center
  call leds.top(0,0,0)
  call leds.bottom.left(1,0,0)
  call leds.bottom.right(1,0,0)
```

Utiliser des événements distincts a l'avantage d'être plus facile à lire et à comprendre. Néanmoins, dans les cas de figure suivants, l'utilisation de l'événement `buttons` est nécessaire : (a) pour pouvoir faire la différence entre un bouton touché et un bouton relâché et (b) pour détecter que deux boutons sont touché en même temps :

```
onevent buttons
  # Allumez les LEDs du haut lorsque le bouton avant est relâché
  when button.forward == 0 do
    call leds.top(32,0,0)
  end

  # Allumez les LEDs du bas lorsque à la fois
  # les boutons gauche et droite sont pressés
  when button.left == 1 and button.right == 1 do
    call leds.bottom.left(0,32,0)
    call leds.bottom.right(0,32,0)
  end
```

Une autre différence est que les événements spécifiques à un bouton ont lieu lorsqu'un bouton est touché ou relâché, tandis que l'événement commun à tous les boutons est déclenché à une fréquence de 20 Hz, à chaque fois que le tableau des variables bouton est mis à jour (voir page 75 pour une explication de ces concepts).

Structures `if`

AESL supporte deux structures différentes :

```
when v == 1 do ... instructions ... end

if v == 1 then ... instructions ... end
```

qui ont des significations différentes (« if ... then » signifie « si ... alors », tandis que nous avons déjà vu que « when ... do » signifie « quand ... exécute ») :

> *quand* la valeur de v *devient* 1, exécute les instructions
> *si* la valeur de v *est* 1, exécute les instructions

Les structures `when` s'utilisent en général pour des variables qui représentent des événements parce que souvent on souhaite réagir quand un événement est déclenché et non pas simplement quand une variable a une certaine valeur. On pourrait réagir à un événement `buttons` en utilisant une structure `if` :

```
onevent buttons
  if button.forward == 1 then
    ... instructions ...
  end
```

Néanmoins, si nous gardons le bouton avant appuyé pendant assez longtemps, ces instructions seraient exécutées plusieurs fois. Si les instructions changent la couleur des LEDs, on ne voit aucune différence, mais dans certains cas, il y a une différence et la structure `when` est alors nécessaire.

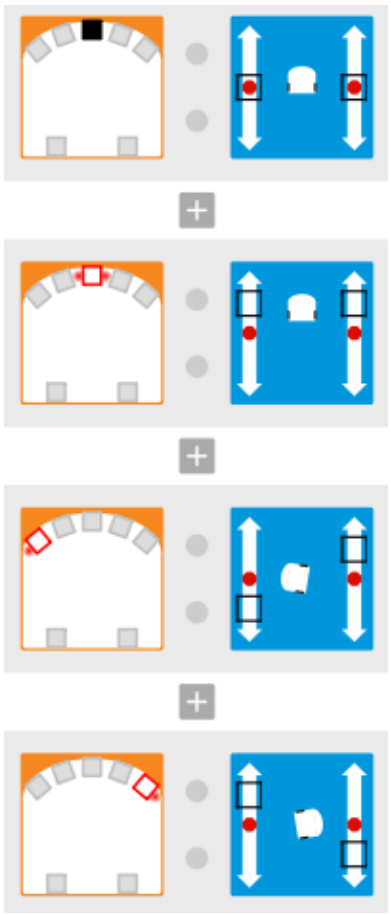
Une structure `if` est plus appropriée lorsque ce qui nous intéresse est la valeur d'une variable et non le changement de cette valeur. Les instructions suivantes attribuent la valeur mesurée maximale entre les deux capteurs arrières à la variable `max` :

```
if prox.horizontal[5] > prox.horizontal[6] then
  max = prox.horizontal[5]
else
  max = prox.horizontal[6]
end
```

Nous verrons des exemples supplémentaires de structures `if` dans la section suivante et dans la Figure 21.3.

Les tableaux

Le programme `likes.aesl` dans le Chapitre 4 du tutoriel VPL permet au robot de suivre votre main alors que vous la bougez d'un côté à l'autre de l'avant du robot, en face des capteurs de proximité avants. Lorsqu'aucun objet n'est détecté, le robot s'arrête ; lorsqu'un objet est détecté devant le capteur central, le robot avance ; lorsqu'un objet est détecté devant le capteur tout à gauche ou tout à droite, le robot tourne dans cette direction.



```
onevent prox
  when prox.horizontal[2] < 1000 do
    motor.left.target = 0
    motor.right.target = 0
  end
  when prox.horizontal[2] > 2000 do
    motor.left.target = 300
    motor.right.target = 300
  end
  when prox.horizontal[0] > 2000 do
    motor.left.target = -300
    motor.right.target = 300
  end
  when prox.horizontal[4] > 2000 do
    motor.left.target = 300
    motor.right.target = -300
  end
```

FIGURE 21.2 – The `likes` program in VPL and AESL

La structure du programme AESL (Figure 21.2) contient un traitement d'événements avec plusieurs structures when. Les valeurs de la position des *sliders* des blocs moteurs sont affectés aux variables moteur correspondantes `motor.left.target` et `motor.right.target` (qui signifient respectivement « moteur.gauche.cible» et « moteur.droit.cible»). Dans VPL, les *sliders* permettent de modifier les valeurs des variables moteur par incréments de 50, mais AESL vous permet d'affecter toutes les valeurs dans l'intervalle -500 à 500.

L'événement est appelé `prox` (une forme abrégée pour « proximité »). Contrairement aux événements bouton qui sont déclenchés lorsque quelque chose « se produit », cet événement est déclenché *10 fois par seconde*. Avant que l'événement ne se produise, des valeurs qui dépendent de ce qui est détecté par les capteurs sont affectées aux variables `prox.horizontal`.

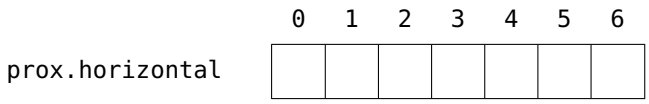
Voir la documentation de l'interface de programmation Thymio pour plus de détails. [1]

Des tableaux comme variables multiples

Le robot Thymio a sept capteurs de proximité horizontaux, 5 à l'avant et 2 à l'arrière. Pour lire les valeurs mesurées par les capteurs, on pourrait définir 7 variables différentes :

```
prox.horizontal.front.0
prox.horizontal.front.1
prox.horizontal.front.2
prox.horizontal.front.3
prox.horizontal.front.4
prox.horizontal.back.0
prox.horizontal.back.1
```

Mais au lieu de faire ainsi, AESL permet de définir des *tableaux*, des suites de variables qui ont toutes le même nom. Les différentes variables sont identifiées avec un nombre pour pouvoir les différencier. Un tableau pour les capteurs de proximité horizontaux est prédéfini est porte le nom `prox.horizontal` :



Les 5 premières entrées du tableau correspondent aux capteurs avant, de gauche à droite. Les 2 dernières correspondent aux capteurs arrières, de gauche à droite. Si vous ne vous rappelez plus de la numérotation, vous pouvez toujours la retrouver dans la documentatoin de l'interface de programmation Thymio. Encore mieux, vous la trouvez aussi sur le diagramme de la petite carte de référence.

Pour accéder à une entrée particulière du tableau, écrivez son numéro entre crochets à la fin du nom du tableau. On appelle ce nombre son *indice* dans le tableau. L'instruction suivante indique que les variables moteur seront mises à 300 lorsque la valeur du *capteur avant central* (indice 2) devient supérieure à 2000 :

```
when prox.horizontal[2] > 2000 do
  motor.left.target = 300
  motor.right.target = 300
end
```

Nous verrons plus tard que l'on peut affecter des valeurs à des variables tableau :

```
timer.period[0] = 1979
```

Les boucles `for` et les variables d'indices*

Une généralisation naturelle des tableaux consiste en l'utilisation d'une variable au lieu du constante comme indice. [2] Le programme `cats.aesl` contient l'instruction suivante :

1. L'unité de mesure pour la *fréquence*, le nombre de fois que quelque chose se produit par seconde est le *hertz*, que l'on abrège *Hz*. La documentation de l'interface précise que l'événemnet `prox` se produit à une fréquence de *10 Hz*.

2. Ceci n'est pas utilisé dans les programmes VPL traduits en AESL, à part dans un cas complexe de création de sons ; c'est pourquoi l'exemple ici est repris des projets AESL.

```
var i

for i in 0:4 do
  if prox.horizontal[i] > DETECTION then
    state = 2
  end
end
```

Précédemment, nous avons utilisé uniquement les variables comprises dans l'interface Thymio ; ici, la première ligne *déclare* une nouvelle variable appelée `i`. L'instruction suivante est une boucle `for` (de l'anglais « pour ». Le mot-clé `in` signifie « dans ») dont la signification est :

— Affecter les valeurs 0, 1, 2, 3, 4 à tour de rôle à la variable `i` ;
— Pour chacune des affectations, exécuter les actions entre `do` (« faire » en anglais) et `end` (« fin » en anglais).

Ici, nous n'avons qu'une seule structure `if` entre `do` et `end`. Elle vérifie la valeur des capteurs de proximité centraux et affecte la valeur 2 à la variable `state` si la valeur obtenue d'un des capteurs est plus grande que la constante `DETECTION`.[3]

La variable `i` prend à tour de rôle les valeurs 0, 1, 2, 3, 4. Ainsi, chaque fois que la structure `if` est exécutée, `prox.horizontal[i]` retourne la valeur mesurée par un capteur différent. Les valeurs de tous les capteurs de gauche à droite sont ainsi lues. Le résultat de la boucle `for` est donc d'affecter la valeur 2 à la variable `state` *si un parmi* les capteurs avants détecte un objet.

Déclarer un tableau

Un tableau de variables se déclare en donnant sa taille entre crochet après le nom du tableau. La taille peut aussi être donnée implicitement en donnant une valeur initiale au tableau :[4]

```
var state[4] # Un tableau avec quatre éléments
var state[] = [0,0,0,0] # Un tableau avec quatre éléments
```

Lors de la traduction du programme VPL, la taille et la valeur initiale est donnée. Le code suivant est correct à condition que le nombre de valeur est égal à la taille indiquée :

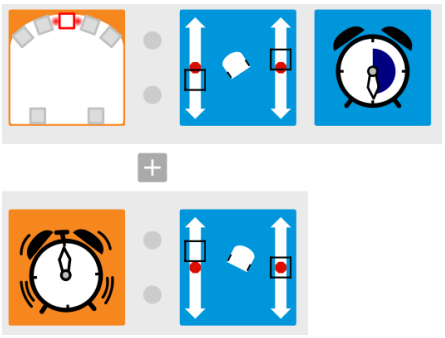
```
var state[4] = [0,0,0,0]
```

3. Se référer à la documentation de l'environnement Aseba Studio pour les instructions sur comment définir des constantes.

4. Un commentaire ne commence pas forcément au début d'une ligne. Tout caractère entre le caractère # et la fin de la ligne est considéré comme un commentaire et est ignoré par l'ordinateur.

Les minuteurs

Le Chapitre 7 a présenté les *minuteurs*. Le programme `shy.aesl` demande au robot de tourner à gauche quand le capteur avant central détecte votre main ; deux secondes plus tard, il tourne à droite :



```
onevent prox
  when prox.horizontal[2] > 2000 do
    motor.left.target = -150
    motor.right.target = 100
    timer.period[0] = 2000
  end

onevent timer0
  timer.period[0] = 0
  motor.left.target = 200
  motor.right.target = 0
```

Le robot Thymio a deux minuteurs. Vous pouvez changer la durée d'un minuteur en affectant une valeur aux entrées 0 ou 1 du tableau `timer.period`. La valeur est en *millisecondes*, c'est-à-dire en millièmes de secondes. Pour régler le minuteur 0 sur 2 secondes, la valeur 2000 (millisecondes) doit être affectée à `timer.period[0]`.

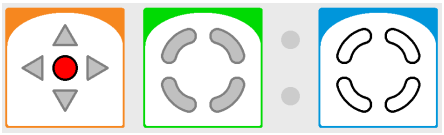
Il y a deux événements, `timer0` et `timer1` (« timer » signifie « minuteur » en anglais), un pour chacun des minuteurs. Lorsque la durée est écoulée, l'événement *timer* est déclenché. Dans le traitement d'événement `timer0`, on remet la durée du timer à 0 pour que l'événement ne se redéclenche plus et on change la vitesse des moteurs.

Dans l'initialisation, le programme règle le minuteur sur 0 pour éviter qu'événement ne se déclenche par accident au début du programme :

```
# stop timer 0
timer.period[0] = 0
```

États

Les Chapitres 8 et 9 ont montré comment utiliser les *états*. Le robot Thymio peut être dans 16 états différents et on peut spécifier dans quel état précis le robot doit être pour qu'un certain événement déclenche une certaine action. Dans le programme `count-to-two.aesl` du Chapitre 9, l'état est mis à 0 lorsque le bouton central est touché. Il compte ensuite si le nombre de frappes des mains est pair ou impair en altérnant entre l'état 0 et l'état 1. Le programme VPL et le traitement d'événement AESL pour le bouton central sont : [5]



```
var state[] = [0,0,0,0]

onevent buttons
  when button.center == 1 do
    state[0] = 0
    state[1] = 0
    state[2] = 0
    state[3] = 0
  end
```

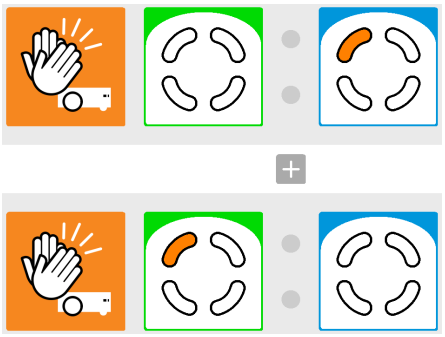
L'état mémorisé dans le tableau `state[]` contient 4 éléments. Chaque élément peut prendre les valeurs 0 ou 1. Il y a donc $2 \times 2 \times 2 \times 2 = 16$ différentes valeurs possible pour ce tableau. On initialise ces éléments du tableau à 0 en affectant les 4 valeurs `[0,0,0,0]`. La valeur initiale du tableau est aussi utilisée pour déduire le nombre d'éléments du tableau ; comme il y a quatre valeurs dans `[0,0,0,0]`, il y aura quatre éléments dans le tableau.

Le bloc événement état (en vert, à côté du bloc événement bouton) a les quatres quartiers en gris ; cela signifie que l'événement aura lieu indépendamment de l'état actuel. Ainsi, à chaque fois que le bouton central est touché, le bloc événement action (en bleu) change l'état actuel à 0 pour tous les éléments du tableau `state`, comme indiqué par les quartiers blancs. Dans le programme AESL correspondant, la structure `when` vérifie si le bouton central a été touché, mais ne vérifie pas les valeurs contenues dans le tableau `state`. Si le bouton central a été touché, chaque élément du tableau est remis à 0.

Il y a deux blocs événement frapper des mains, associés à des blocs état différents. Dans le programme textuel, un seul traitement d'événement `mic` est utilisé. Les instructions à exécuter seront déterminées à l'aide de structures `if` (Figure 21.3).

La signification du mot-clé `and` (« et » en anglais) est que *toutes* les conditions dans la structure `if` doivent être vérifiées pour que les instructions entre le `then` et le `end` soient exécutées. Si tous les éléments du tableau `state` sont 0, on affecte la valeur 1 à l'élément `state[0]`, tandis que l'on affecte la valeur 0 aux autres éléments. Ceci correspond à un bloc événement état avec tous les quartiers blancs et à un bloc action état avec le quartier supérieur gauche orange et les autres blancs. De même, si la valeur de `state[0]` est 1 et les valeurs des autres éléments est 0, les valeurs de tous les éléments de `state` sont remis à 0.

5. Le programme AESL montré ici est différent de celui généré par VPL pour des raisons qui seront expliquées plus bas dans ce chapitre.



```
onevent mic
  if state[0] == 0 and
     state[1] == 0 and
     state[2] == 0 and
     state[3] == 0 then
    state[0] = 1
    state[1] = 0
    state[2] = 0
    state[3] = 0
  end
  if state[0] == 1 and
     state[1] == 0 and
     state[2] == 0 and
     state[3] == 0 then
    state[0] = 0
    state[1] = 0
    state[2] = 0
    state[3] = 0
  end
```

FIGURE 21.3 – Réagir aux frappes des mains selon l'état actuel

Sous-routines

Il arrive souvent qu'une même suite d'instructions doive être exécutée à plusieurs reprises à des endroits différents du programme. On pourrait écrire ces instructions une fois puis les copier chaque fois qu'on en a besoin. Une solution plus simple consiste à utiliser une *sous-routine*, qui permet d'attribuer un nom à une suite d'instructions. Dans ce programme, l'expression `sub display_state` déclare une sous-routine qui attribue au nom `display_state` une suite d'instructions, ici, une seul instruction, `call leds.circle` :

```
# subroutine to display the current state
sub display_state
  call leds.circle(
    0, state[1]*32, 0, state[3]*32, 0, state[2]*32, 0, state[0]*32)
```

Lorsque la sous-routine est *appelée*, la suite d'instructions attribuée au nom de la sous-routine est exécutée :

```
callsub display_state
```

La fonction d'interface `leds.circle` permet de contrôler les huit LEDs entourant les boutons. Là il faut vraiment vous référer au diagramme de la petite carte de référence pour apprendre quel paramètre correspond à quelle LED !

L'intensité de chaque LED peut être contrôlée en donnant une valeur entre 0 (LED éteinte) et 32 (LED complètement allumée). Les LEDs avant, arrière, gauche et droite sont mises à 0 (éteintes), tandis que les LEDs diagonales sont allumées si l'élément correspondant du tableau d'état est 1 et éteintes s'il est 0. Pour obtenir ce résultat, on utilise les *expressions arithmétiques* `state[...]*32` qui multiplient la valeur des éléments du tableau par 32. Si une des valeurs est 0, le résultat sera 0, tandis que si elle vaut 1, le résultat sera 32.

Fonctions natives

Le programme ci-dessus a un problème. Puisque les valeurs des éléments du tableau `state` sont affectées une à une, il est possible qu'un événement différent se produise alors que les valeurs d'une partie des éléments ont été modifiées, mais pas de tous. Pour modifier tous les éléments en même temps, il faut d'abord mettre les nouvelles valeurs dans un nouveau tableau `new_state` et puis les copier dans le premier tableau `state` :

```
# variables for state
var state[4] = [0,0,0,0]
var new_state[4] = [0,0,0,0]

onevent buttons
  when button.center == 1 do
    new_state[0] = 0
    new_state[1] = 0
    new_state[2] = 0
    new_state[3] = 0
  end

  call math.copy(state, new_state)
  callsub display_state
```

La *fonction native* `math.copy` est utilisée pour copier des tableaux. Les fonctions natives sont déjà intégrées dans le robot Thymio et sont plus efficaces que des suites d'instructions dans AESL. Les fonctions natives sont décrites dans la documentation d'Aseba.

La version actuelle d'AESL autorise les affectations de tableaux entiers, de telle sorte qu'il aurait été possible d'utiliser une instruction d'affectation : [6]

```
  state = new_state
```

6. L'affectation de tableaux se traduit en fait par une suite d'affectations individuelles, élément par élément, de sorte qu'il n'y a aucun d'intérêt à utiliser une affectation de tableau.

Cinquième partie

Annexes

Annexe A

L'interface d'utilisateur VPL

Tout en haut de la fenêtre VPL se trouve une barre d'outils :

Nouveau : Supprime le programme actuel et vide la zone de programmation.

Ouvrir : Ouvre un projet existant dans VPL. Une fenêtre va s'ouvrir pour que vous choisissiez le fichier (extension `aesl`) à ouvrir.

Sauvegarder : Sauvegarde le programme actuel. Il est conseillé de cliquer souvent sur ce bouton pour ne pas perdre votre travail si un plantage devait survenir.

Sauvegarder sous : Sauvegarde le programme actuel sous un *nom différent*. Utilisez ce bouton lorsque vous avez un programme et que vous souhaitez pouvoir le modifier sans perdre la version actuelle du programme.

Annuler : Annule la dernière action comme par exemple la suppression d'une paire événement-actions.

Rétablir : Rétablit la dernière action qui a été annulée.

Lancer : Lance le programme actuel. Ce bouton n'est actif que si la compilation a réussi. Si vous modifiez le programme après l'avoir déjà lancé une fois, le bouton clignotera pour vous rappeler que vous devez cliquer dessus pour charger dans le Thymio la version modifiée du programme.

Arrêter : Arrête le programme qui est en train d'être exécuté et arrête les moteurs. Utilisez ce bouton quand votre programme a mis les moteurs du robot en mouvement mais ne possède pas de paire événement-actions pour les arrêter.

Mode avancé : Le mode avancé offre des fonctionnalités supplémentaires : les états, les minuteurs, les accéléromètres et la possibilité de changer les seuils des capteurs.

Mode débutant : L'icône ci-dessus devient en mode avancé. Cliquez sur ce bouton pour revenir en mode débutant.

Aide : Affiche la documentation VPL dans votre navigateur internet. Une connexion internet est nécessaire. Vous trouverez aussi la documentation sous https://www.thymio.org/fr:thymiovpl.

Exporter : Exporte une image graphique du programme vers un fichier. Vous pouvez ensuite importer cette image dans un document comme un livre de cours ou une fiche de travail. Plusieurs formats sont disponibles. Les SVG offrent la meilleure qualité, mais le format PNG est plus souvent reconnu.

Annexe B

Résumé des blocs VPL

Blocs événement

Les boutons. Cliquez sur un ou plusieurs boutons ; ils deviendront rouge. Un événement sera déclenché chaque fois que les boutons rouges sont touchés.

Capteurs horizontaux (cinq à l'avant du Thymio et deux à l'arrière). Cliquez sur un ou plusieurs des petits carrés : ils changeront de couleur. Initiallement, tous les carrés sont gris, ce qui signifie que les mesures de tous les capteurs seront ignorées.

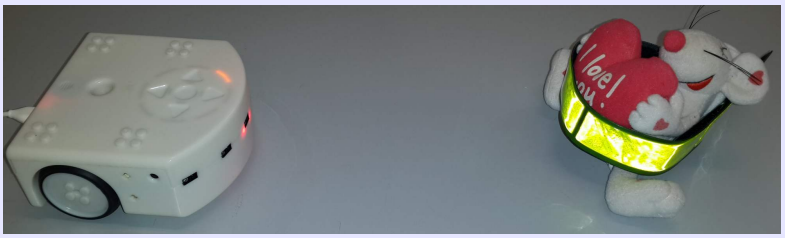
Si le carré est blanc avec un bord rouge , un événement est déclenché si beaucoup de lumière est réfléchie.

Si le carré est noir , un événement est déclenché si peu de lumière est réfléchie.

Truc

Les objets ordinaires doivent être très proches du Thymio pour pouvoir être détectés par les capteurs horizontaux. Vous pouvez augmenter de beaucoup la portée des capteurs en collant du *ruban adhésif réflecteur* comme on en utilise sur les vélos sur les objets. [a]

Comparez l'image suivante avec la Figure 8.3 :



a. Je remercie Francesco Mondada pour m'avoir donné ce truc !

Capteurs du sol (deux sur le dessous du Thymio). Ce bloc s'utilise de la même façon que les capteurs horizontaux.

Capteurs, mode avancé. Ces blocs s'utilisent comme les blocs précédents. Le *slider*

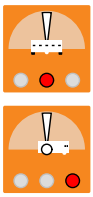
du haut change le seuil au-dessus duquel un objet est détecté et le *slider* du bas change le seuil au-dessous duquel l'absence d'un objet est détectée.

Il y a un mode additionnel (le carré est gris foncé) dans lequel un événement est déclenché lorsque la valeur mesurée est entre le seuil supérieur et le seuil inférieur.

Tape. Un événement est déclenché lorsqu'on donne une petite tape au Thymio.

Tape, mode avancé. Ce bloc s'utilise comme le bloc précédent. Cliquez sur le petit cercle au centre ou à droite pour passer à l'événement accéléromètre.

Accéléromètre, mode avancé. Déplacez l'angle blanc dans le demi-cercle vers la droite ou la gauche. Un événement sera déclenché lorsque l'angle droite/gauche, respectivement avant/arrière du Thymio est à l'intérieur de cet intervalle.



Événement état, mode avancé. L'événement est déclenché seulement si les différents quartiers de l'état actuel correspondent aux quartiers orange et blancs de ce bloc. Les quartiers de l'état actuel correspondant aux quartiers gris de ce bloc ne doivent pas forcément être de la même couleur.

Blocs action

Moteurs. Déplacez les *sliders* gauche et droite vers le haut pour augmenter la rotation en avant des moteurs gauche et droit. Déplacez ces *sliders* vers le bas pour augmenter la rotation en arrière des moteurs gauche et droit.

Lumières du haut. Déplacez les trois *sliders* vers la droite pour augmenter l'intensité de rouge, de vert, respectivement de bleu des lumières du haut.

Lumières du bas. Permet d'allumer les lumières du bas. Ce bloc s'utilise comme le bloc précédent.

Musique. Les six petits ronds représentent des notes de musique. Un rond noir est une

note courte tandis qu'un rond blanc est une note longue. Cliquez sur un cercle pour changer sa longueur. Les cinq barres horizontales représentent les différentes hauteurs de notes disponibles. Cliquez sur une des barres pour placer une note sur cette barre.

Minuteur, mode avancé Le minuteur peut être reglé pour une durée allant jusqu'à quatre secondes. Cliquez où vous voulez sur le cercle blanc qui représente le cadran d'un réveil. Il y aura alors une courte animation et la durée jusqu'à l'alarme sera coloré en bleu.

États, mode avancé Les quatre quartiers du bloc correspondent aux quatre quartiers d'un état. Cliquez sur un des quartiers pour le mettre en gris, orange ou blanc.

Remarques concernant les blocs VPL

Tourner sur place ou doucement

Lorsque les moteurs tournent à la même vitesse dans des sens différents, le robot tourne sur place. Si, au contraire, un seul moteur est allumé, le robot ira à la fois vers l'avant et tournera vers le côté opposé à celui du moteur allumé. Il vous faudra peut-être mettre plus de puissance dans les moteurs pour compenser le frottement du sol.

Événements capteurs rapidement répétés

Dans la plupart des projets, on attribue une couleur (blanc, noir ou gris foncé) à un carré dans un bloc capteur pour indiquer que le capteur correspondant joue un rôle pour déterminer quand les actions associées seront exécutées. Cepandant, si vous laissez tous les capteurs dans leur couleur grise originale, alors aucune condition ou restriciton n'est imposée. L'événement sera alors déclenché 10 fois par secondes, peu importe les valeurs mesurées par les capteurs.

Événements boutons rapidement répétés

La même remarque vaut pour les événements boutons, avec la différence que ceux-ci sont déclenchés 20 fois par seconde.

Annexe C

Quelques trucs pour programmer avec VPL

Explorer et expérimenter

Comprendre chaque bloc événement et action Pour chaque bloc événement et chaque bloc action, prenez-vous le temps de faire plusieurs essais jusqu'à ce que vous compreniez exactement comment il fonctionne. Pour mieux comprendre comment un bloc action fonctionne, créez une paire avec un bloc événement bouton. C'est un événement facile qui vous permettra de bien discerner l'action du bloc action. Pour les blocs événement, vous pouvez construire une paire avec l'action changer de couleur.

Tester les blocs événement capteurs Les petites lumières rouges à côté de chaque capteur permettent de voir quand ce capteur détecte un objet. Déplacez par exemple vos doigts devant les capteurs et regardez quelles lumières sont allumées, ce qui indique quels capteurs détectent vos doigts. Créez une paire événement-actions qui consiste en un événement capteur et l'action couleur du haut et jouez avec les différents réglages des petits carrés dans le bloc événement (gris, blanc, noir et gris foncé en mode avancé).

Construisez un programme

Faites un plan pour votre programme Avant d'écrire un programme, commencez par rédiger une description du fonctionnement voulu de votre programme : une phrase pour chaque paire événement-actions.

Construisez une paire événement-actions à la fois Une fois que vous aurez compris comment chacune des paires événement-actions fonctionnent, vous pourrez les mettre ensemble pour créer votre programme.

Testez chaque nouveauté de votre programme Testez votre programme à chaque fois que vous ajoutez une nouvelle paire événement-actions pour que vous puissiez rapidement trouver la paire à l'origine d'une erreur dans votre programme.

Utilisez Sauvegarder sous quand vous avez changé votre programme Avant de modifier votre programme, cliquez sur  pour sauvegarder votre programme en utilisant un autre nom. Si votre modification devait ne pas marcher, il vous sera alors facile de revenir à la version précédente.

Montrez ce qui se passe Utilisez des couleurs pour montrer ce que le programme est en train de faire. Par exemple, si un capteur d'une paire a une action associée tourner à gauche et un capteur d'une autre paire a une action associée tourner à droite, ajoutez une action aux deux paires qui affichent des couleurs différentes. Vous pourrez ainsi

voir si le problème est dû aux capteurs ou si ce sont les moteurs qui ne réagissent pas correctement aux événements capteurs.

Régler les problèmes

Utilisez une surface lisse Vérifiez que votre Thymio se déplace sur une surface lisse et propre. Les moteurs risquent sinon de ne pas réussir à déplacer le robot, ou alors les virages risquent de ne pas être réguliers.

Utilisez un long cable Assurez-vous d'avoir un cable assez long. Si le robot va loin, le cable pourrait sinon freiner ou arrêter le robot.

Les événements capteurs peuvent ne pas être déclanchés Les événements capteurs sont déclanchés 10 fois par seconde. Si le robot se déplace très rapidement, il est possible qu'un événement ne soit pas déclanché.

Par exemple, si le robot est censé s'arrêter lorsqu'il détecte le bord de la table mais qu'il avance très rapidement, il est possible qu'il tombe de la table avant que l'événement capteurs n'arrête les moteurs. Lorsque vous lancer un programme, commencez avec une vitesse lente. Vous pouvez ensuite l'augmenter graduellement.

Comme autre exemple, considérez le programme du Chapitre 5 où Thymio suivait une ligne. L'algorithme de ce programme repose sur la capacité à détecter le moment où un des capteurs voit la ligne et l'autre ne la voit plus. Si le robot avance trop rapidement, le moment où seul un capteur détecte la ligne est trop bref pour déclancher un événement.

Problèmes avec les capteurs du bas Dans les programmes comme celui à peine mentionné, le capteur doit pouvoir distinguer beaucoup de lumière réfléchie de peu de lumière réfléchie. Veillez à avoir un grand contraste entre les deux. Si votre table, par exemple, n'est pas assez claire, fixer des feuilles blanches permettra d'obtenir de meilleurs résultats.

Vous pouvez sinon aussi ajuster les seuils de détection en mode avancé.

Les paires événement-actions sont exécutées à la suite En théorie, les paires événement-actions sont exécutées *simultanément*—en même temps ; en pratique, elles sont exécutées à la suite, dans l'ordre dans lequel elles apparaissent dans votre programme. Comme on peut le voir dans l'exercice 4.2, ceci peut poser problème car la seconde action peut entrer en conflit avec ce qui a été exécuté par la première action.

Problèmes avec l'événement frapper des mains *N'utilisez pas* l'événement frapper des mains lorsque les moteurs tournent. Les moteurs génèrent beaucoup de bruit et peuvent provoquer des événement frapper des mains indésirables.

De même, *n'utilisez pas* l'événement tappe et l'événement frapper des mains dans le même programme. En effet, donner une tappe au robot génère du bruit que le robot peut interpréter comme une frappe des mains.

Annexe D

Astuces pour utiliser les *sliders*

Régler les sliders dans les blocs moteurs

Il est difficile de régler précisément les *sliders*, par exemple pour que les deux moteurs tournent à la même vitesse. Pour améliorer la précision, nous pouvons utiliser la traduction des paires événement-actions dans le programme textuel AESL.

Truc

Vous remarquerez que déplacer les *sliders* des blocs action moteurs change la vitesse cible des moteurs (`motor.X.target`) par palliers de 50, entre −500 et 500. En déplaçant délicatement ces *sliders*, vous pouvez choisir toutes ces valeurs comme vitesses.

La Figure D.1 montre le programme de la Figure 4.4 (celle où votre animal de companie vous aime et vous suit partout) ainsi que la traduction textuelle, visible sur la droite de la fenêtre VPL. Ce texte change automatiquement quand vous déplacez les *sliders*.

`onevent prox` signifie : chaque fois que l'événement de l'échantillonage des capteurs horizontaux (les capteurs de *proximité*, ici abbregé *prox*) est déclanché, les instructions jusqu'au `end` suivant seront exécutées. L'événement proximité se produit 10 fois par seconde.

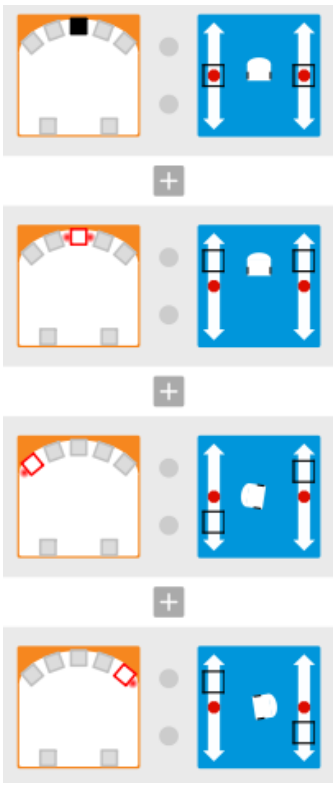
Lorsqu'un événement se produit, les valeurs des capteurs sont verifiées grâce aux structures de condition. Le capteur numéro 2 (avant central) est verifié en premier : `prox.horizontal[2]`. Si cette valeur est inférieure à 1000, les vitesses des moteurs gauches et droits sont mises à 0 grâce aux instructions :

```
motor.left.target = 0
motor.right.target = 0
```

Chaque structure `if ... then ... end` vérifie un capteur spécifique et exécute les instructions associées si le résultat du test est vrai. Le programme exécute en fait l'algorithme suivant :

0. Vérifie que rien ne soit en face ; si c'est le cas, le robot s'arrête.
1. Vérifie si quelque chose est en face ; si c'est le cas, le robot avance.
2. Vérifie s'il y a quelque chose à gauche ; si oui, tourne à gauche.
3. Vérifie s'il y a quelque chose à droite ; si oui, tourne à droite.

Une fois que les valeurs de tous les capteurs ont été lues et que l'action appropriée a été exécutée, le programme attend le prochain événement `prox` et recommence les tests. Ceci se répéte tant que le programme n'est pas arrêté.



```
onevent prox
    if prox.horizontal[2] < 1000 then
        motor.left.target = 0
        motor.right.target = 0
    end
    if prox.horizontal[2] > 2000 then
        motor.left.target = 300
        motor.right.target = 300
    end
    if prox.horizontal[0] > 2000 then
        motor.left.target = -300
        motor.right.target = 300
    end
    if prox.horizontal[4] > 2000 then
        motor.left.target = 300
        motor.right.target = -300
    end
```

FIGURE D.1 – Un programme VPL et le programme textuel correspondant.

Régler la durée du minuteur

Truc

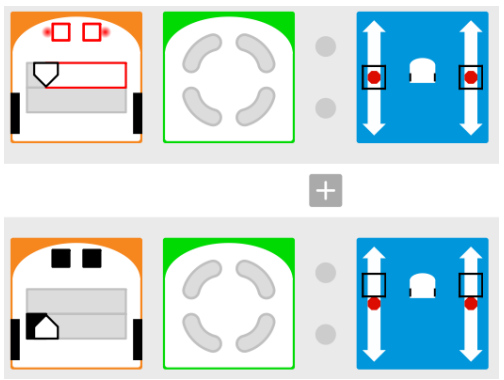
La durée du bloc action minuteur peut être réglé par multiples de quarts de seconde (250 millisecondes) jusqu'à quatre secondes.

Le bloc événement capteurs en mode avancé

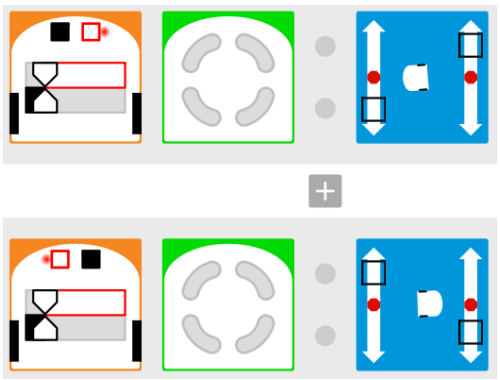
Cette section décrit les fonctionnalités des blocs événement capteurs disponibles en mode avancé.

Régler les seuils des capteurs

En mode débutant, les seuils des capteurs sont fixés. Pour les capteurs horizontaux, une valeur supérieure à 2000 signifie que beaucoup de lumière est réfléchie et un événement sera déclanché si le carré correspondant est blanc, tandis qu'une valeur inférieure à 1000 signifie que peu de lumière est réfléchie et un événement sera déclanché si le carré correspondant est noir. Pour les capteurs du bas, les seuils sont 450 et 400.



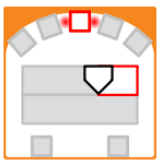
(a) Démarrer et arrêter le robot



(b) Corriger quand le robot dévie

FIGURE D.2 – Un programme pour suivre une ligne

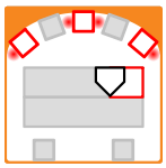
En mode avancé, les seuils peuvent être réglés. Le *slider* du haut permet de régler le seuil au-dessus duquel un événement blanc se produit et le *slider* du bas permet de régler le seuil au-dessous duquel un événement noir se produit :



La Figure D.2 montre le programme pour suivre une ligne (Figure 5.2) en mode avancé. Les *sliders* ont été réglés de sorte à obtenir des seuils très faibles : 100 à la fois pour le seuil supérieur et inférieur.

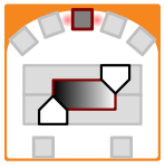
Plusieurs capteurs

Si plusieurs capteurs sont activés dans un bloc événement, ils partagent les mêmes seuils :



Déclancher des événements pour des valeurs entre deux seuils

Il existe un mode additionnel représenté par un carré gris foncé :



Dans ce mode, un événement se produit si la valeur mesurée est supérieure au seuil inférieur (réglé par le *slider* inférieur) et inférieure au seuil supérieur (réglé par le *slider* supérieur).