

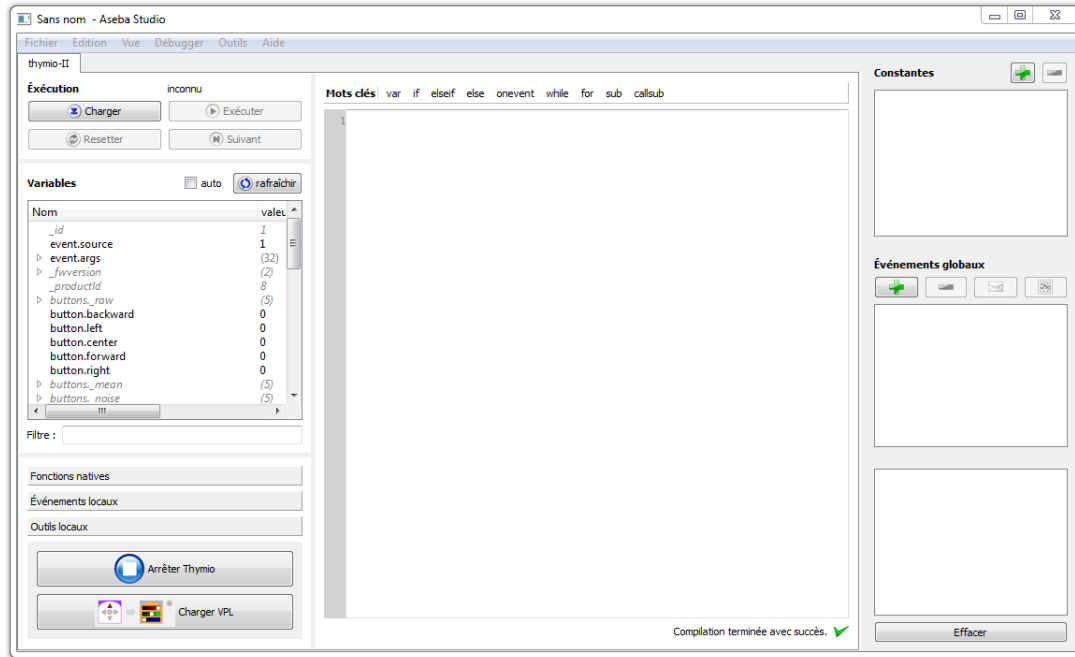
PROGRAMMER THYMIO EN MODE TEXTE

Démarrer Aseba Studio pour Thymio

Branchez un Thymio à votre ordinateur, puis cliquez sur l'icône Aseba Studio pour Thymio :

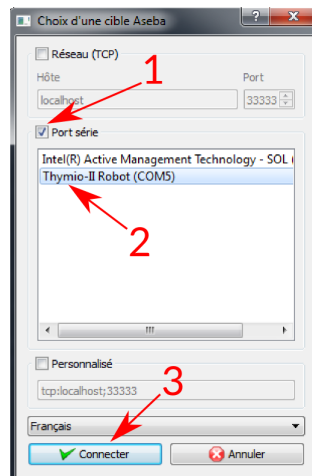


pour lancer Aseba Studio, le logiciel utilisé pour programmer le Thymio en mode texte:



Si Thymio n'est pas branché ?

Si le Thymio n'est pas branché, ou si vous lancez Aseba Studio (et non pas la version « pour Thymio »), une fenêtre de sélection du robot s'ouvrira :



Branchez le Thymio et cliquez sur :

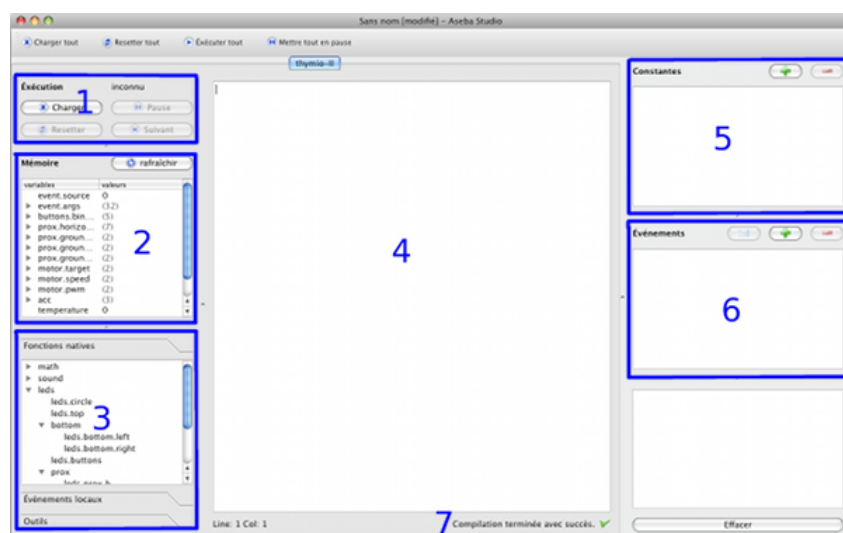
1. Port Série (c'est l'USB)
2. Thymio-II Robot
3. Connecter

La différence entre Aseba Studio et Aseba Studio pour Thymio est simplement que la deuxième icône ouvrira Studio directement, sans afficher la fenêtre de sélection de robot, si un Thymio est branché à l'ordinateur.

Premiers pas avec Aseba et Thymio

En appuyant sur le bouton du milieu, on allume Thymio. En appuyant ensuite sur les flèches, Thymio change de couleur : en réappuyant sur le bouton du milieu on peut activer un des modes pré-programmés. Dans la suite de l'exercice on va créer notre propre mode, donc ajouter un choix dans les couleurs.

Connecter Thymio à l'ordinateur avec le câble USB et lancer Aseba Studio. Voici à quoi ressemble Aseba Studio :



1. Ces boutons servent à charger et exécuter le code qu'on a écrit.
2. Cette fenêtre donne une vision d'ensemble de l'état des différents éléments du robot : capteurs, actuateurs, variables.
3. Ici on trouve les fonctions et les événements disponibles qu'on peut utiliser dans le code
4. Au milieu on va écrire notre *code* qui va déterminer le comportement du robot.
5. Ici on peut déterminer des constantes.
6. Ici on peut ajouter nos événements.
7. Cette ligne nous indique si nos instructions sont complètes et correctement écrites.

Les capteurs

On peut utiliser l'interface de Aseba studio pour voir la valeur des *capteurs* du robot, dans le menu sur le côté. Un capteur sert à mesurer quelque chose de physique, comme de la lumière, du son ou une accélération, et à le traduire en une valeur numérique, que le robot peut comprendre et utiliser. Par exemple les *capteurs de distance* devant le robot permettent de mesurer la distance vers les objets proches. C'est la valeur **prox.horizontal** dans la fenêtre **Variables**. Si on met la main devant le robot et que la case **auto** est cochée, on peut voir la valeur changer en direct.

Nom	valeurs	Nom	valeurs
button.right	0	button.right	0
prox.horizontal	(7)	prox.horizontal	(7)
0	0	0	4293
1	0	1	3863
2	0	2	3355
3	0	3	3313
4	0	4	0
5	0	5	0
6	0	6	0
prox.ground.ambient	(2)	prox.ground.ambient	(2)
prox.ground.reflected	(2)	prox.ground.reflected	(2)
prox.ground.delta	(2)	prox.ground.delta	(2)
motor.left.target	0	motor.left.target	0
motor.right.target	0	motor.right.target	0
motor.left.speed	0	motor.left.speed	0

Dans la première image on voit que les capteurs ne détectent rien (0). Dans la deuxième, on a mis la main devant, les valeurs sont élevées.

Qu'est-ce que le code ?

C'est une série d'*instructions* qu'on donne au robot, qu'il va *exécuter* et qui vont définir son comportement. Il faut écrire le code avec des mots clés bien précis, que le robot peut comprendre; on ne peut pas simplement lui parler en français.

Une astuce pour écrire le code plus vite

Dans Aseba Studio, on peut tirer les mots clés depuis les menus de côté jusque dans la fenêtre principale. Dans le cas de blocs de code très similaires, on peut aussi les copier (Ctrl-C) et coller (Ctrl-V) sous Windows et Linux (ou (Commande-C) et (Commande-V) sous Mac).

Les commentaires

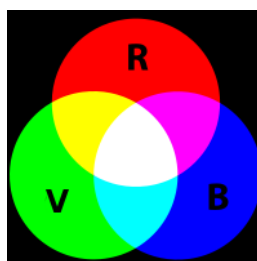
Ce qu'on écrit dans le code est interprété par le robot comme des instructions. Cependant, parfois on peut vouloir écrire des indications pour nous-mêmes. Dans ce cas, on va faire un commentaire. Le signe # permet d'indiquer que la partie qui suit n'est pas une instruction pour le robot. On verra un exemple plus tard.

Dans cette partie, on va programmer Thymio pour qu'il change de couleur quand on appuie sur les boutons.

Les LEDs

Une *LED* ([Diode électroluminescente](#)) est un composant qui permet de faire de la lumière sur le robot. Une LED a une couleur déterminée par sa fabrication. Par exemple, il y a des LEDs rouges devant le robot, des vertes dessus etc.

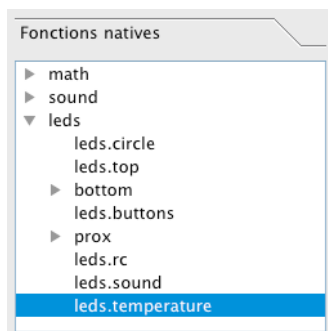
On a vu que le robot pouvait changer de couleur. Comment ça se passe ? En réalité, il y a trois LEDs différentes l'une à côté de l'autre, très proches. Une rouge, une verte, une bleue. Ces trois couleurs sont les couleurs primaires (additives) ; on peut créer les autres en les mélangeant. Par exemple si on allume bleu et rouge en même temps, on obtient magenta. On appelle cela une *LED RVB* (pour rouge, vert, bleu).



Allumons une LED

Pour pouvoir allumer une lumière nous-mêmes sur le robot, on va commencer à *coder*, donc à écrire des ordres pour le robot.

Si on clique sur **Fonctions natives** -> **leds** on peut voir une liste de *fonctions* (une fonction est une des instructions possibles pour le robot). Ces fonctions servent à allumer les différentes LEDs. Par exemple **leds.top** (top signifie haut) sert à allumer la LED RVB du haut.



Dans la fenêtre principale, tape ceci :

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

call doit être utilisé pour appeler une fonction; **leds.top** sert à allumer les LEDs RVB du dessus du robot.

Si on laisse le pointeur de la souris sur le nom de la fonction, on verra apparaître des informations pour pouvoir l'utiliser. Entre les parenthèses on donne des indications à la fonction : **des arguments**.



Dans notre cas il y en a 3 : les valeurs des trois parties de la LED : rouge, vert et bleu. La valeur 0 signifie éteint, et 32, allumé au maximum. On peut faire une lumière moins forte avec une valeur entre 0 et 32.

Maintenant clique sur **Charger**, puis **Exécuter**. Le robot va exécuter le code. On voit alors les LEDs du dessus allumées en bleu.

Si on change le code pour mettre :

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

la LED sera alors rouge. De même :

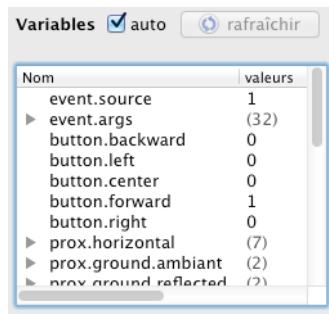
```
call leds.top(0,32,32)
```

donnera une LED turquoise. Vous pouvez essayer de jouer sur les couleurs en changeant ces valeurs.

Les conditions

Pour l'instant, dès qu'on appuie sur **Exécuter**, le robot exécute notre code (allumer une LED). On peut avoir envie de lier ça par exemple avec un bouton, pour allumer une LED quand on appuie.

Pour détecter l'appui des boutons, on peut vérifier la variable dans la fenêtre **Variables** : **button.nom**



Ici on voit que le bouton avant (forward) est appuyé.

Dans ce cas, on va utiliser la *condition* **if ... then ... else ... end** (si ... alors ... sinon ... fin). L'idée est de pouvoir dire au robot **si** le bouton est appuyé, **alors** allume une led. Dans Aseba, ça se fait avec la syntaxe suivante :

```
if button.forward==1 then # si le bouton de devant est appuyé
  call leds.top(0,0,32) # alors on allume en bleu
else
  call leds.top(0,0,0) # sinon on éteint
end
```

ce code dit que si on appuie sur le bouton de devant (**button.forward**), on allume la LED de dessus en bleu, sinon, on l'éteint. On peut noter les phrases après les # qui ne sont pas en langage *Aseba* mais en français. Ce sont des commentaires, des notes pour la personne qui code, pas pour le robot, donc on les met à part avec le signe #.

Ce code est-il suffisant pour obtenir le comportement voulu ?

Non. Avec ce code, si on appuie et relâche le bouton, le robot ne change pas de couleur. On peut voir que ça ne marche qu'une seule fois, au démarrage. Le robot fait juste une fois les instructions qu'on lui a donné.

Si l'on veut qu'à chaque fois qu'on appuie sur le bouton, ça allume la LED, il faut vérifier **régulièrement** si le bouton est appuyé. Pour ça, on va utiliser les *événements*.

Les événements

Un événement, c'est un moment déterminé par certaines conditions, auquel on peut exécuter du code. Un événement peut avoir lieu plusieurs fois! Par exemple si on regarde dans notre interface en bas, il y a une liste d'**Événements locaux**.



L'événement **buttons** correspond à chaque fois qu'on vérifie si un bouton est appuyé. De même, **buttons.forward** correspond à "chaque fois qu'on appuie sur le bouton de devant". Donc maintenant on ajoute cette ligne avant notre code:

```
# à chaque fois qu'on vérifie les bouton
onevent button.forward
if button.forward==1 then # si le bouton de devant est appuyé
  call leds.top(0,0,32) # alors on allume en bleu
else
  call leds.top(0,0,0) # sinon on éteint
end
```

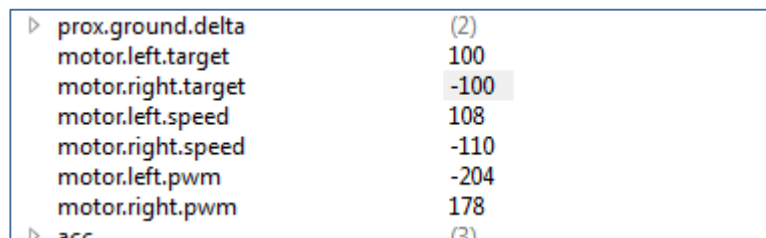
Si l'on teste ce code (**Charger, Exécuter**) on voit que le robot allume la lumière bleue dès qu'on met le doigt sur le bouton, et l'éteint dès qu'on l'enlève.

Il y a d'autres événements disponibles : **motor** pour les moteurs, **acc** pour l'accéléromètre etc.

Maintenant on va faire bouger le robot. Selon le bouton qu'on touche, le robot changera de direction.

Les moteurs

Pour les moteurs on voit dans la mémoire qu'on a 3 variables différentes. Pour le moment on va utiliser les deux premières. **motor.coté.target** nous permet de dire quelle vitesse on veut avoir. **motor.coté.speed** nous dit quelle est la vitesse actuelle.



▷ prox.ground.delta	(2)
motor.left.target	100
motor.right.target	-100
motor.left.speed	108
motor.right.speed	-110
motor.left.pwm	-204
motor.right.pwm	178
▷ acc	(3)

On voit que la vitesse réelle (**motor.coté.speed**) est un peu différente de la vitesse demandée (**motor.coté.target**)

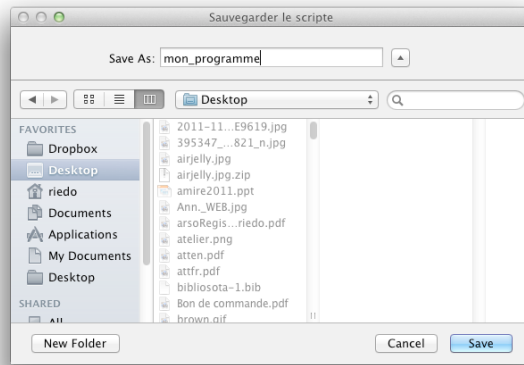
On peut maintenant écrire un code qui active les moteurs en fonction des boutons.

Vous pouvez maintenant essayer de commander le robot avec les boutons et de le faire bouger (n'oubliez pas de cliquer sur **Charger**, puis **Exécuter**).

Et si on veut débrancher le robot ?

Tester le robot avec le câble n'est pas très pratique. Alors comment faire ?

Si on débranche le robot de l'ordinateur, Aseba Studio va essayer de se reconnecter, et quand on branchera à nouveau le robot, on pourra continuer à programmer. Néanmoins, si l'on quitte Aseba Studio, le programme sera perdu. Il vaut mieux donc sauvegarder le code avant de débrancher le robot. Pour ce faire, aller dans le menu **Fichier** et clic sur **Sauvegarder**. Une boîte de dialogue comme celle-ci devrait apparaître :



Ensuite le comportement reste et on peut jouer avec le robot. Si quoi que ce soit arrive à Aseba Studio pendant que l'on joue avec le robot, on pourra redémarrer Aseba Studio et recharger le fichier qu'on a sauvegardé.

Utiliser les capteurs : s'arrêter au bord de la table

Maintenant on va utiliser les capteurs de proximité pour éviter que le robot ne tombe de la table. Il y a 9 capteurs sur chaque Thymio : 5 devant, 2 dessous, 2 derrière. On va utiliser ceux de dessous pour s'arrêter en bord de table.

Le capteur de proximité

Un capteur de proximité peut mesurer les distances aux objets proches. Pour le faire il utilise deux composants : un émetteur de lumière infrarouge, et un récepteur. L'émetteur envoie de la lumière infrarouge (invisible pour nous) et le récepteur mesure combien de lumière revient. Si un objet est proche, beaucoup de lumière va être reflétée dessus et revenir vers le robot. S'il est loin, moins de lumière reviendra. On peut donc mesurer la distance aux objets.

Dans notre cas, on peut utiliser les capteurs dessous pour détecter la table. Laisser le robot sur la table et regarder dans l'interface, la valeur de **prox.ground.delta**.

▼ prox.ground.delta	(2)
0	574
1	631

Maintenant recommençons en soulevant le robot. On voit que la valeur change beaucoup.

▼ prox.ground.delta	(2)
0	7
1	8

Quand le robot est sur la table, les valeurs sont autour de 600 (beaucoup de lumière revient). Lorsqu'on le soulève, les valeurs sont beaucoup plus petites (peu de lumière). On va donc ajouter en bas de tout notre code ceci :

27. `onevent prox`

28. `# si un capteur mesure une valeur plus petite que 300, donc il ne voit pas la table`

29. `if prox.ground.delta[0]<300 or prox.ground.delta[1]<300 then`

- 30. # on arrête les moteurs
- 31. motor.left.target=0
- 32. motor.right.target=0
- 33. call leds.top(32, 0, 0) # on allume le robot en rouge pour signaler l'arrêt d'urgence
- 34. else
- 35. call leds.top(0, 0, 0) # si on ne détecte pas de bord de table, on éteint la lumière d'urgence
- 36. end

Si la valeur d'un des deux capteurs est basse (pas de table détectée) on arrête le robot. Avec ce programme, on peut voir que le robot s'arrête en bord de table et s'allume en rouge (ça marche également si on soulève le robot).

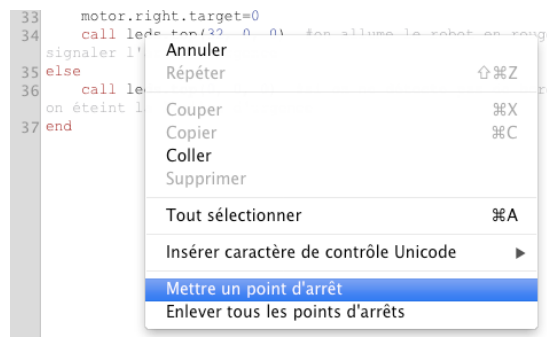
Le débogueur

Il peut arriver que pendant qu'on teste le robot, on ait envie de voir ce qui se passe. On peut le faire pour corriger des erreurs par exemple, ou pour exécuter les instructions pas à pas. Dans notre cas on va voir ce qui se passe quand on arrive en bord de table.

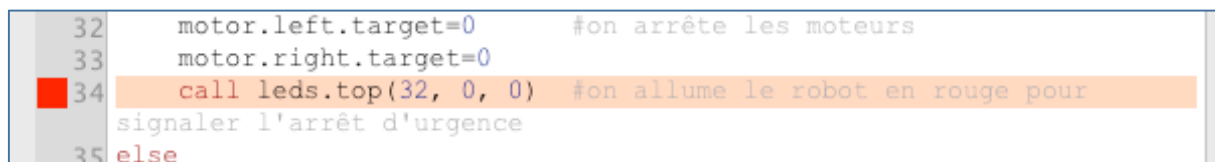
Si on clique avec le bouton droit sur la ligne

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

On peut sélectionner *Mettre un point d'arrêt*.



Le point d'arrêt est signalé par un carré rouge.



cela signifie que quand on arrive à cet endroit, le robot va attendre qu'on lui dise de continuer.

Faisons un essai. Il faut décocher la case **auto** si on veut garder la valeur au moment du point d'arrêt. Une fois le point d'arrêt mis, on fait avancer le robot jusqu'au bord de la table. Il s'arrête; on va pouvoir observer la situation à ce moment, par exemple ici on voit que les capteurs du bas ont mesuré les valeurs 151 et 45 (peu de lumière, donc la table n'est plus détectée).

prox.ground.delta	(2)	32	motor.left.target=0	#o
0	151	33	motor.right.target=0	
1	45	34	call leds.top(32, 0, 0)	#o
motor.left.target	0	35	signaler l'arrêt d'urgence	
			else	

Dès qu'on rappelle sur exécuter, le robot continue (il s'allume en rouge) jusqu'au prochain point d'arrêt. On peut enlever les points d'arrêts en cliquant avec le bouton droit sur la ligne et en sélectionnant *Enlever le point d'arrêt*.

L'accéléromètre

Comment peut-on détecter un choc ? Quel capteur permet de faire cela ?

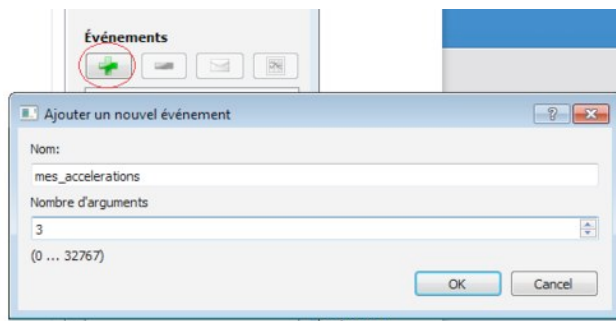
La détection de chocs est très courante et utile. Voyons quelques exemples :

- dans une voiture, on peut détecter un choc en cas d'accident pour déclencher l'airbag.
- dans de nombreux téléphones portables, on peut détecter si l'utilisateur tape avec le doigt, une ou plusieurs fois de suite.
- dans un ordinateur portable, on détecte les chocs pour protéger le disque dur.

Ce genre d'événement est détecté grâce à l'accéléromètre. L'accéléromètre permet de mesurer des accélérations, dont par exemple la gravité. On peut aussi mesurer la chute libre et les chocs.

Observer l'activité de l'accéléromètre : réagir aux chocs et émettre un son

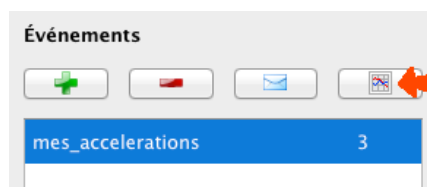
Aseba Studio permet de faire des graphiques de variables. Pour ça, il faut créer notre propre *événement*. Si on veut observer les trois axes de l'accéléromètre (on mesure l'accélération en trois dimensions) il faut créer un événement avec 3 arguments. Pour ça, on clique sur le + pour ajouter un événement, on lui donne un nom (p.ex **mes_accelerations**) et on spécifie le nombre d'arguments (3).



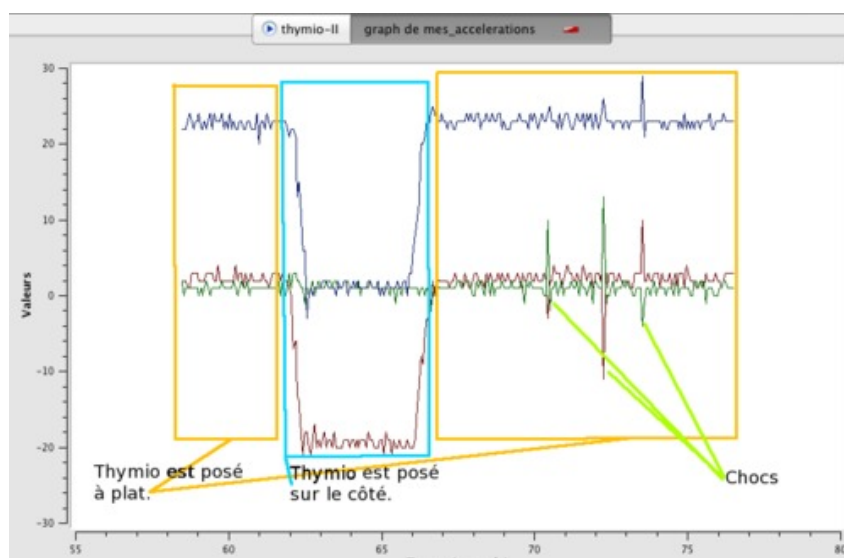
Maintenant, dans le code, on peut envoyer la valeur de l'accélération (variable **acc** dans la fenêtre **Variables**, qui comporte 3 éléments) lors de l'événement **acc**, à notre nouvel événement **mes_accelerations** qui peut transmettre 3 arguments:

```
onevent acc
emit mes_accelerations acc
```

Ensuite, dans la fenêtre **Événements**, on peut sélectionner **mes_accelerations** et cliquer sur le bouton de graphique.



Un nouvel onglet s'ouvre avec un graphique des 3 accélérations. On peut alors observer ce qui se passe si on tourne le robot ou qu'on tape dessus.



Ici on voit les 3 axes en 3 couleurs différentes. Quand le robot est posé sur ses roues, on détecte la gravité sur l'axe Z (en bleu). Quand on le met sur le côté, on détecte l'accélération sur un autre axe (le rouge). En cas de choc, on voit des pics.

Utiliser l'accéléromètre

On peut utiliser l'accéléromètre de différentes façons. Comme on l'a vu, la variable **acc** donne la valeur des accélérations sur les trois axes, et l'événement **acc** a lieu à chaque fois que les valeurs d'accélérations sont mesurées.

Il y a en plus un événement **tap** qui a lieu lorsque le robot détecte une tape ou un choc. On peut ainsi utiliser cet événement pour faire faire du bruit au robot lorsqu'on le tape.

```
onevent tap  
  call sound.system(4) # cette fonction permet de jouer un son système
```