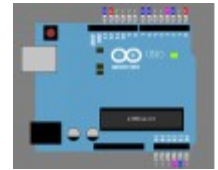


# UnoArduSimV2.0.1 Aide Complète



## Table des Matières

### Aperçu

#### Volet de Code, Préférences et Éditer / Examiner

##### Volet de Code

##### Préférences

##### Éditer / Examiner

#### Volet de Variables et fenêtre Modifier / Surveiller la Variable

#### Volet du Banc de Laboratoire

##### Le 'Uno'

##### Dispositifs 'I / O'

##### 'Serial' Monitor ('SERIAL')

##### Software Serial ('SFTSER')

##### Lecteur de Disque SD ('SD\_DRV')

##### Esclave de Registre de Décalage ('SRSLV')

##### Esclave SPI configurable ('SPISLV')

##### Esclave I2C ('I2CSLV')

##### Moteur Pas à Pas ('STEPR')

##### Moteur DC ('MOTOR')

##### ServoMoteur ('SERVO')

##### Pulseur Binaire ('PULSER')

##### Un-Tir ('1SHOT')

##### Générateur de Fonction Analogique ('FUNCGEN')

##### Haut-parleur Piézoélectrique ('PIEZO')

##### Bouton Poussoir ('PUSH')

##### Résistance de l'interrupteur à Glissière ('R = 1K')

##### DEL Colorée ('LED')

##### Curseur Analogique

### Menus

#### Fichier:

Charger INO ou PDE Prog ( ctrl-L )

Éditer / Examiner (ctrl-E )

Enregistrer

Enregistrer Sous

Suivant ( '#inclure' )

Précédent

Quitter

#### Trouver:

Trouver la Fonction/Var Suivante

Trouver la Fonction/Var Précédente

Définir le Texte de Recherche (ctrl-F)

Trouver le Texte suivant

Trouver le Texte précédent

#### Exécuter:

Un Pas Dans (F4)

Un Pas Franchir (F5)

Un Pas Sortir (F6)

Exécuter Vers (F7)

Exécuter Jusqu'à (F8)

Exécuter (F9)

Arrêter (F10)

Réinitialisez

Animer

Ralenti

#### Options:

[Enjamber des Structors / Opérateurs](#)  
[Modélisation de Registre-Allocation](#)  
[Erreur sur non-Initialisé](#)  
['loop\(\)' Delay ajouté](#)

#### **Configurer:**

[Dispositifs 'I / O'](#)  
[Préférences](#)

#### **VarRafrachir :**

[Autoriser la Contraction automatique \(-\)](#)  
[Autoriser la Réduction](#)  
[Minimal](#)  
[Modifications de HighLight](#)

#### **Fenêtres:**

['Serial' Monitor](#)  
[Tout restaurer](#)  
[Formes d'Onde Numériques](#)  
[Forme d'Onde Analogique](#)

#### **Aidez-moi:**

[Fichier d'aide rapide](#)  
[Fichier d'aide complet](#)  
[Corrections de bugs](#)  
[Changement / Améliorations](#)  
[Sur](#)

#### **La Modélisation**

[Carte 'Uno' et Les Dispositifs 'I / O'](#)  
[Synchronisation](#)  
[Dispositifs 'I / O'](#)  
[Des Sons](#)

#### **Limitations et éléments non pris en charge**

[Fichiers Inclus](#)  
[Allocation de mémoire dynamique et RAM](#)  
[Allocations de mémoire 'Flash'](#)  
['String' Variables](#)  
[Bibliothèques Arduino](#)  
[Pointeurs](#)  
[Objets 'class' et 'struct'](#)  
[Portée](#)  
[Qualificateurs 'unsigned' , 'const' , 'volatile' , 'static'](#)  
[Directives du Compilateur](#)  
[Éléments de Langage Arduino](#)  
[C / C ++ - Éléments de Langage](#)  
[Modèles de Fonction](#)  
[Émulation en Temps Réel](#)

#### **Notes de Version**

##### **Corrections de Bugs**

[V2.0.1- Janv. 2018](#)  
[V2.0- Déc. 2017](#)  
[V1.7.2- fév.2017](#)  
[V1.7.1- Fév.2017](#)  
[V1.7.0- Déc.2016](#)  
[V1.6.3- septembre 2016](#)  
[V1.6.2- septembre 2016](#)  
[V1.6.1- août 2016](#)  
[V1.6- Juin 2016](#)  
[V1.5.1- Juin 2016](#)  
[V1.5 - Mai 2016](#)  
[V1.4.3 - Avril 2016](#)  
[V1.4.2 - mars 2016](#)  
[V1.4.1 - Janv. 2016](#)  
[V1.4 - Déc. 2015](#)

[V1.3 - Oct. 2015](#)

[V1.2 - juin 2015](#)

[V1.1 - mars 2015](#)

[V1.0.2 - août 2014](#)

[V1.0.1 - Juin 2014](#)

### **Changements / Améliorations**

[V2.0.1- Janv. 2018](#)

[V2.0 Déc. 2017](#)

[V1.7.2- février 2017](#)

[V1.7.1- février 2017](#)

[V1.7.0- Déc. 2016](#)

[V1.6.3- septembre 2016](#)

[V1.6.2- septembre 2016](#)

[V1.6.1- août 2016](#)

[V1.6 - Juin 2016](#)

[V1.5.1 - Juin 2016](#)

[V1.5 - Mai 2016](#)

[V1.4.2 - mars 2016](#)

[V1.4 - décembre 2015](#)

[V1.3 - oct 2015](#)

[Version 1.2 Juin 2015](#)

[V1.1 - mars 2015](#)

[V1.0.1 - Juin 2014](#)

## Aperçu

UnoArduSim est un outil de simulation freeware en temps **réel** (voir Modélisation pour les **restrictions de temps**) que j'ai développé pour les étudiants et les passionnés d'Arduino. Il est conçu pour vous permettre d'expérimenter et de déboguer facilement des programmes Arduino **sans avoir besoin de matériel réel**. Il est destiné à la carte **l'Arduino 'Uno'** et vous permet de choisir parmi un ensemble de virtuel 'I / O dispositifs', et de configurer et de connecter ces dispositifs à votre virtuelle 'Uno' dans le **Volet du Banc de Laboratoire**. - vous n'avez pas besoin de vous soucier des erreurs de câblage, des connexions brisées / lâches ou des dispositifs défectueux qui perturbent le développement et les tests de votre programme.

UnoArduSim fournit des messages d'erreur simples pour toutes les erreurs d'analyse ou d'exécution qu'il rencontre et permet le débogage avec **Réinitialiser**, **Exécuter**, **Exécuter-Vers**, **Exécuter-Jusqu'à**, **Arrêtez** et les opérations **Avancer d'un Pas** flexibles dans le **Volet de Code**, avec une vue simultanée de toutes les variables locales, tableaux et objets globaux et actifs dans le **Volet de Variables**. La vérification des limites des tableaux d'exécution est fournie, et un débordement RAM ATmega sera détecté (et la ligne du programme coupable sera mise en évidence!). Les conflits électriques avec les dispositifs 'I / O' attachés sont signalés et signalés lorsqu'ils se produisent.

Lorsqu'un fichier programme INO ou PDE est ouvert, il est chargé dans le **Volet de Code** du programme. Le programme est ensuite analysé, et "compilé" dans un exécutable tokenized qui est alors prêt pour l' **exécution simulée** (contrairement à Arduino.exe, un exécutable binaire autonome n'est *pas* créé) Toute erreur d'analyse est détectée et signalée en mettant en surbrillance la ligne qui a échoué et en signalant l'erreur sur la **Barre d'État** tout en bas de la fenêtre de l'application UnoArduSim. Une **fenêtre Éditer / Examiner peut être ouverte pour vous permettre de voir et de modifier une version en surbrillance de la syntaxe de votre programme utilisateur**. Les erreurs au cours de l'exécution simulée (telles que les vitesses de transmission en désaccord) sont signalées dans la barre d'état et via une boîte de message contextuelle.

UnoArduSim V2.0 est une implémentation pratiquement complète de l' **Arduino Programming Language V1.6.6** **comme documenté au [arduino.cc](http://arduino.cc)**. Langue page Web de référence, et avec des ajouts comme indiqué dans la version Téléchargement page Release Notes. Bien que UnoArduSim ne prenne pas en charge l'implémentation C ++ complète du compilateur GNU sous-jacent Arduino.exe, il est probable que seuls les programmeurs les plus avancés trouveront un élément C / C ++ qu'ils souhaitent utiliser (et bien sûr, il y a toujours des codage des solutions de rechange pour ces fonctionnalités manquantes). En général, je n'ai pris en charge que les fonctionnalités C / C ++ les plus utiles pour les amateurs et les étudiants Arduino. Par exemple, **'enum'** et **'#define'** sont pris en charge, mais les pointeurs de fonction ne le sont pas. Même si les objets définis par l'utilisateur ( **'class'** et **'struct'** ) et (la plupart) les surcharges d'opérateurs sont pris en charge, *l'héritage multiple ne l'est pas*.

Comme UnoArduSim est un simulateur de langage de haut niveau, **seules les instructions C / C ++ sont prises en charge**, *les instructions de langage d'assemblage ne le sont pas*. De même, parce que ce n'est pas une simulation de machine à faible niveau, les **registres ATmega328 ne sont pas accessibles à votre programme** soit pour la lecture ou l'écriture, bien que l'allocation de registre, passant et le retour sont émulées (il vous choisissent que dans le menu **Options**).

À partir de la version 2.0, UnoArduSim a intégré la prise en charge automatique d'un sous-ensemble limité des bibliothèques fournies par Arduino, à savoir: **'Stepper.h'**, **'SD.h'**, **'Servo.h'**, **'SoftwareSerial.h'**, **'SPI.h'**, **'Wire.h'** et **'EEPROM.h'** (version 2). Pour toute bibliothèque **'#include'** de bibliothèques créées par l'utilisateur, UnoArduSim **ne** recherche **pas** la structure de répertoire d'installation habituelle d'Arduino pour localiser la bibliothèque; à la place, vous **devez** copier l'en-tête correspondant (".h") et le fichier source (".cpp") dans le même répertoire que le fichier programme sur lequel vous travaillez (sous réserve bien sûr de la limitation du contenu de tout **'#include'** fichier doit être entièrement compréhensible pour l'analyseur UnoArduSim).

J'ai développé UnoArduSimV2.0 dans QtCreator avec un support multilingue, et il est actuellement disponible uniquement pour Windows <sup>™</sup>. Porting to Linux ou MacOS, c'est un projet pour le futur. UnoArduSim est né des

simulateurs que j'ai développés au fil des ans pour les cours que j'ai enseignés à l'Université Queen's, et il a été testé assez largement, mais il y a forcément quelques bogues qui s'y cachent encore. Si vous souhaitez signaler un bug, veuillez le décrire (brièvement) dans un email à [unoArduSim@gmail.com](mailto:unoArduSim@gmail.com) et **assurez-vous de joindre votre code source Arduino** pour que je puisse répliquer le bug et le réparer. Je ne répondrai pas aux rapports de bugs individuels, et je n'ai pas de délais garantis pour les correctifs dans une version ultérieure (souvenez-vous qu'il y a presque toujours des solutions de contournement!).

À votre santé,

Stan Simmons, Ph. D, P. Eng.  
Professeur agrégé (retraité)  
Département de génie électrique et informatique  
Queen's University  
Kingston, Ontario, Canada

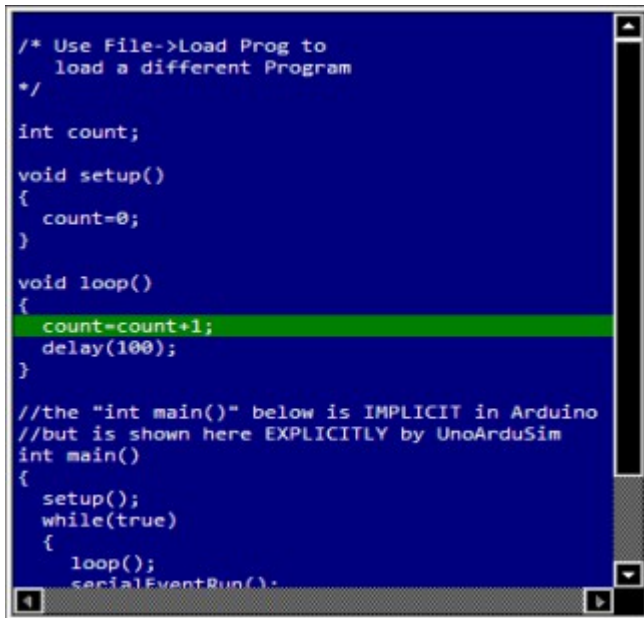


## Volet de Code, Préférences et Éditer / Examiner

(A part: les exemples de fenêtres présentés ci-dessous sont tous sous un thème de couleur Windows-OS choisi par l'utilisateur qui a une couleur de fond de fenêtre bleu foncé)

### Volet de Code

Le **Volet de Code** affiche votre programme utilisateur et met en surbrillance le suivi de son exécution. Une fois qu'un programme chargé est analysé avec succès, la première ligne '**main()**' est mise en surbrillance et le programme est prêt à être exécuté. Notez que '**main()**' est implicitement ajouté par Arduino (et par UnoArduSim) et vous ne l'incluez **pas** dans le cadre de votre fichier de programme utilisateur. L'exécution est sous le contrôle du menu **Exécuter** et des boutons de la **Barre d'Outils** associés et des raccourcis de la touche de fonction.



```
/* Use File->Load Prog to
load a different Program
*/





int count;

void setup()
{
  count=0;
}

void loop()
{
  count=count+1;
  delay(100);
}


//the "int main()" below is IMPLICIT in Arduino
//but is shown here EXPLICITLY by UnoArduSim
int main()
{
  setup();
  while(true)
  {
    loop();
    serialEventRun();
  }
}
```



Après avoir progressé d'une (ou de plusieurs) instructions (vous pouvez utiliser les boutons de la






**Barre d'Outils** , , , ou ), la ligne de programme qui sera exécutée ensuite est mise en évidence - la ligne en surbrillance est toujours la ligne suivante **prête à être exécutée**.

De même, lorsqu'un programme en cours atteint un point d'arrêt (**Exécuter-Vers** temporaire), l'exécution est interrompue et la ligne de point d'arrêt est mise en surbrillance (et est alors prête à être exécutée).

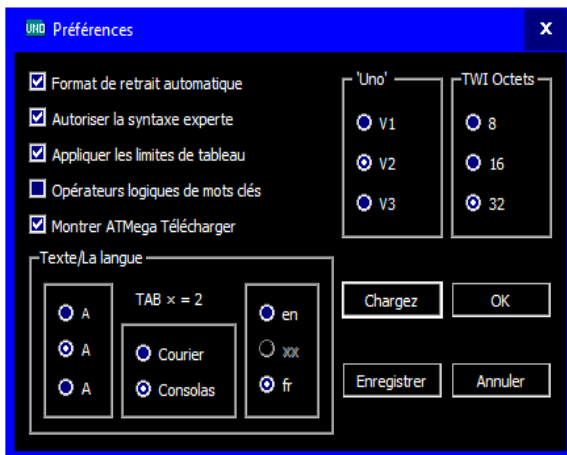
Si l'exécution du programme est actuellement interrompue et que vous cliquez dans la **fenêtre du Volet de Code**, la ligne que vous venez de cliquer est mise en surbrillance. Cela ne change cependant pas la ligne de programme en cours en ce qui concerne l'exécution du programme. Mais vous pouvez faire en sorte que l'exécution *progresses jusqu'à* la ligne

que vous venez de mettre en surbrillance, puis en cliquant sur **Exécuter-Vers**  Bouton de **Barre d'Outils**. Cette fonctionnalité vous permet d'accéder rapidement et facilement à des lignes spécifiques d'un programme, de sorte que vous puissiez ensuite passer ligne par ligne sur une partie du programme qui vous intéresse.

Si votre programme chargé contient des fichiers '**#include**', vous pouvez les déplacer en utilisant **Fichier | Précédent** et **Fichier | Suivant** (avec Boutons de la **Barre d'Outils**  et ).

Au menu vous permet de **passer** rapidement entre les **fonctions** du **Volet de Code** (après la première cliquant sur une ligne à l'intérieur pour donner le focus) à l'aide des commandes **Suivant** et **Précédent** (avec des boutons **barre d'outil**  et  ou les raccourcis clavier **PgDn** et **PgUp**). Alternativement, vous pouvez trouver le texte spécifié avec ses commandes de texte (avec les boutons de la barre d'outils  et , ou les raccourcis clavier vers le haut et vers le bas), après avoir d'abord utilisé **Trouver | Définir le Texte de la Recherche** ou le bouton de la **Barre d'Outils** .

## Préférences

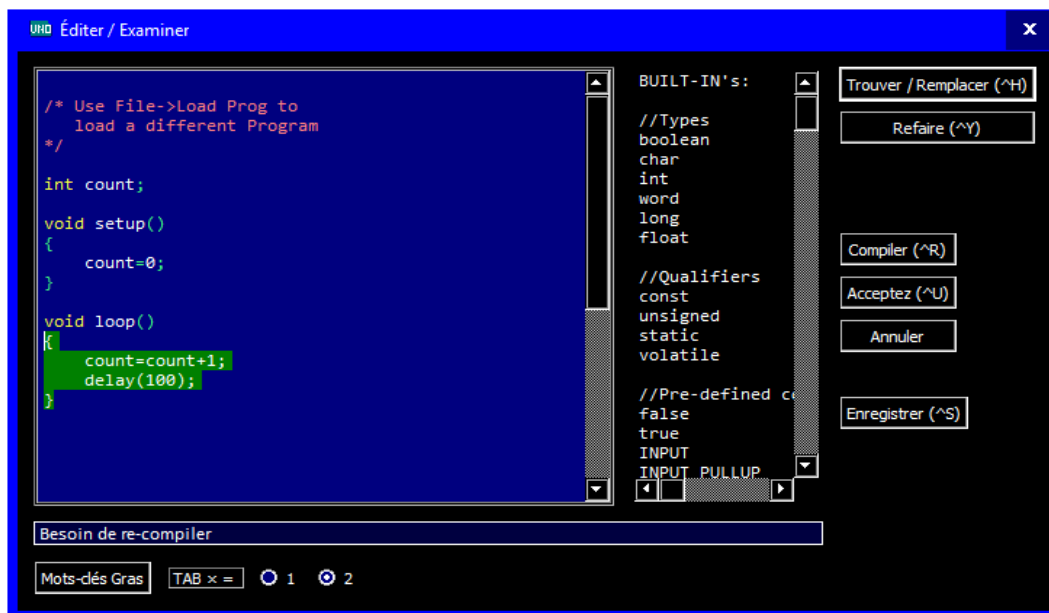


**Configurer | Préférences** permet aux utilisateurs définir les préférences de programme et de visualisation (qu'un utilisateur voudra normalement adopter à la prochaine session). Ceux-ci peuvent donc être sauvegardés et chargés à partir d'un fichier **myArduPrefs.txt** qui réside dans le même répertoire que le programme 'Uno' chargé ( **myArduPrefs.txt** est automatiquement chargé s'il existe)

Cette boîte de dialogue permet de choisir entre deux polices mono-espacées et trois tailles de caractères, ainsi que d'autres préférences diverses. À partir de la version 2.0, le langage chpice est maintenant inclus. - ceci inclut toujours l'anglais ( **en** ), plus une ou deux autres langues locales de l'utilisateur (où elles existent), et une substitution basée sur le code de langue ISO-639 à deux lettres sur la toute première ligne du **myArduPrefs**, fichier **txt** (si on est fourni

là-bas). Les choix n'apparaissent que si un fichier de traduction ".qm" existe dans le dossier de traduction (dans le répertoire de base de UnoArduSim.exe)

## Éditer / Examiner



En double-cliquant sur n'importe quelle ligne dans le **Volet de Code** (ou en utilisant le menu **Fichier/2** ), une **fenêtre EÉditer /// RExaminer** ouverte pour permettre les modifications de votre fichier programme, avec la ligne actuellement sélectionnée dans le **Volet de Code** .

Cette fenêtre a une capacité d'édition complète avec mise en évidence de la syntaxe dynamique (différentes couleurs de surbrillance sont utilisées pour les mots-clés C ++, les commentaires, etc. ). La mise en surbrillance de la syntaxe en gras et la mise en forme automatique du niveau d'indentation sont possibles (en supposant que vous l'ayez sélectionné en utilisant **Configurer | Préférences** ). Vous pouvez également sélectionner des appels de fonction intégrés (ou des '#define' constantes) intégrées à votre programme dans la liste fournie - il suffit de double-cliquer sur l'élément de la liste souhaitée pour l'ajouter à votre programme à la position actuelle du curseur (les types de variable d'appel de fonction sont donnés à titre indicatif et sont supprimés pour laisser place à des espaces fictifs ajoutés à votre programme).



La fenêtre a **Trouver** (utiliser **ctrl-F**) et la fonction **Trouver / Remplacer** (utiliser **ctrl-H**) . La fenêtre **Éditer / Examiner** possède des boutons **Reprendre** ( **ctrl-Z** ) et **Refaire** ( **ctrl-Y** ) (qui apparaissent automatiquement).

Pour annuler **toutes les modifications** effectuées depuis la première ouverture du programme pour modification, cliquez sur le bouton **Annuler** . Pour accepter l'état actuel, cliquez sur le bouton **Accepter** et le programme est automatiquement analysé à nouveau (et téléchargé sur 'Uno' si aucune erreur n'est détectée) et le nouvel état apparaît dans la barre d' **état** principale de UnoArduSim .

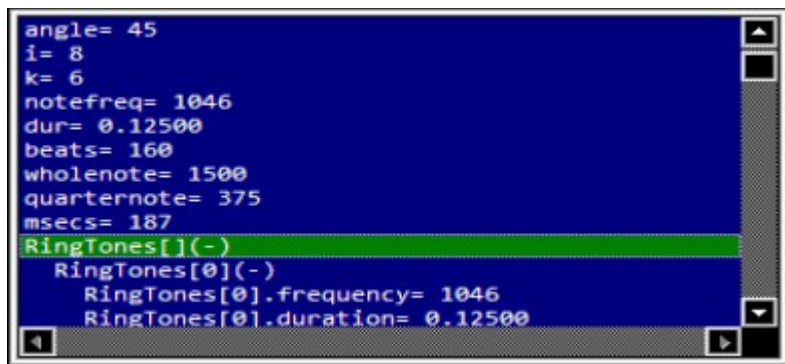
Un bouton **Compiler** ( **ctrl-R** ) (plus une boîte de message d' **état d'analyse** associée comme dans l'image ci-dessus) a été ajouté pour permettre le test des modifications sans avoir à fermer la fenêtre au préalable. Un bouton **Enregistrer** ( **ctrl-S** ) a également été ajouté en tant que raccourci (équivalent à un **Accept** plus un **Enregistrer séparé plus tard** dans la fenêtre principale).

En mode **Annuler** ou **Accepter** sans modification, la ligne actuelle du **Volet de Code** change pour devenir la **dernière position d'insertion / affichage du curseur** , et vous pouvez utiliser cette fonction pour passer le **Volet de Code** sur une ligne spécifique (éventuellement pour préparer une **exécution**). To ), vous pouvez également utiliser **ctrl-PgDn** et **ctrl-PgUp** pour passer au saut de ligne suivante (ou précédent) dans votre programme - ceci est utile pour naviguer rapidement vers des emplacements significatifs (comme des lignes vides entre des fonctions) . Vous pouvez également utiliser **ctrl-Home** et **ctrl-End** pour sauter au début et à la fin du programme, respectivement.

La mise en forme automatique du retrait au niveau de l'onglet est effectuée lorsque la fenêtre s'ouvre, si cette option a été définie sous **Configurer | Préférences**. Vous pouvez également ajouter ou supprimer des onglets vous-même à un groupe de lignes consécutives présélectionnées à l'aide de la touche fléchée **droite** ou **gauche** , mais l' **indentation automatique doit être désactivée** pour éviter de perdre vos propres niveaux d'onglet.

Et pour vous aider à mieux suivre vos contextes et accolades, cliquer sur une ' { ' ou une ' } ' accolade met en évidence tout le texte entre cette accolade et son partenaire correspondant .




## **Volet de Variables et fenêtre Modifier / Surveiller la Variable**





```
angle= 45
i= 8
k= 6
notefreq= 1046
dur= 0.12500
beats= 160
wholenote= 1500
quarternote= 375
msecs= 187
RingTones[0](-)
RingTones[0](-)
RingTones[0].frequency= 1046
RingTones[0].duration= 0.12500
```

Le **vVolet de Variables** se trouve juste en dessous du **vVolet de Code** et affiche les valeurs actuelles de chaque variable locale / active / locale / locale / active / variable dans le programme chargé. Au fur et à mesure que votre exécution de programme se déplace entre les fonctions, **le contenu change pour refléter uniquement les variables locales accessibles à la fonction / portée en cours, plus tout globales déclarés par l'utilisateur** . Toutes les variables

déclarées comme **'const'** ou **'PROGMEM'** (allouées à 'Flash' memory) ont des valeurs qui ne peuvent pas changer, et pour économiser de l'espace, elles ne sont donc *pas affichées* . Les instances **'Servo'** et **'SoftwareSerial'** ne contiennent aucune valeur utile et ne sont donc pas affichées.

Les commandes du menu **Trouver** (avec les boutons de la **barre d' outils**  et  , Ou les raccourcis clavier **PgDn** et **PgUp**) vous permettent de **passer** rapidement entre les **variables** dans les **vaVolet de Variables** après le premier clic sur une ligne à l' intérieur pour donner le focus). Alternativement, vous pouvez trouver le texte spécifié avec ses commandes de recherche de texte (avec les boutons de la barre d' outils  et



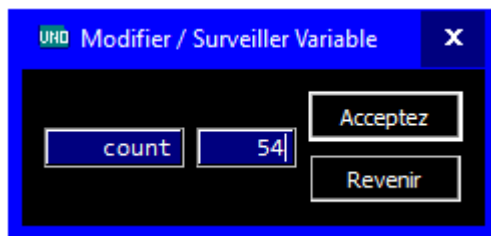
{ , ou les raccourcis clavier vers le haut et vers le bas ), après avoir d'abord utilisé **Trouver | Définir le Texte de Recherche** ou .

Les **tableaux** et les **objets** sont affichés dans un format **non développé** ou **étendu**, avec respectivement un signe plus (+) ou moins (-). Le symbole d'un tableau **x** montre comme **x[]**. Pour le développer afin d'afficher tous les éléments du tableau, il suffit de cliquer une fois sur **x[]** (+) dans le **Volet de Variables**. Pour revenir à une vue non développée, cliquez sur le **[]** (-). La valeur par défaut non étendue pour un objet **p1** montre comme **p1** (+). Pour le développer afin d'afficher tous les membres de cette **instance** '**class**' ou '**struct**', cliquez une fois sur **p1** (+) dans le **Volet de Variables**. Pour revenir à une vue non développée, cliquez une fois sur **p1** (-).

Si vous **cliquez** une **seule** fois **sur une ligne pour** le **sélectionner** (il peut être variable simple, ou l'agrégat (+) ou (-) ligne d'un tableau ou un objet, ou un élément de tableau unique ou membre objet), puis en faisant une **cours** -Till provoquera la reprise et l'arrêt de l'exécution au prochain **accès en écriture** n'importe où dans l'agrégat sélectionné ou à l'emplacement de la variable unique sélectionnée.

Lorsque vous utilisez **Avancer d'un Pas** ou **Exécuter**, les mises à jour des valeurs de variables affichées sont effectuées en fonction des paramètres utilisateur **définis** dans le menu **VarRafraichir** - ceci permet une gamme complète de comportements allant des mises à jour périodiques minimales aux mises à jour immédiates complètes. Les mises à jour réduites ou minimales sont utiles pour réduire la charge du processeur et peuvent être nécessaires pour empêcher l'exécution de tomber en arrière en temps réel sous ce qui serait autrement excessif lors de la mise à jour du **Volet de Variables**. Lorsque **Animate** est **activé**, ou si l'option de menu **Mettez en Surbrillance les Modifications** est sélectionnée, la modification de la valeur d'une variable pendant l'**exécution** entraînera la mise à jour **immédiate** de sa valeur affichée, qui sera mise en surbrillance, ce qui fera défiler le **Volet de Variables** (nécessaire) à la ligne qui contient cette variable, et l'exécution ne sera plus en temps réel!

**Lorsque l'exécution se bloque** après **Avancer d'Un Pas**, **Exécuter-Vers**, **Exécuter-Jusqu'à** ou **Exécuter-puis-Arrêtez**, le **Volet de Variables** met en évidence la variable correspondant aux **emplacements d'adresse modifiés** (le cas échéant) par la **dernière instruction** au cours de cette exécution (y compris par initialisations de déclarations de variables). Si cette instruction a **complètement** rempli un **objet ou un tableau**, la ligne **parentale** (+) ou (-) de cet agrégat est mise en surbrillance. Si, au lieu de cela, l'instruction a modifié un emplacement actuellement visible, il est mis en surbrillance. Mais si le (s) emplacement (s) modifié (s) se cache (nt) actuellement dans un tableau ou un objet non-développé, cette **ligne parent** agrégée reçoit une **police italique en guise** de repère visuel indiquant que quelque chose à l'intérieur a été écrit. provoque la mise en évidence de son **dernier** élément ou membre modifié.



La fenêtre **Modifier / Surveiller Variable** vous donne la **possibilité de suivre n'importe quelle valeur de variable pendant l'exécution**, ou de **changer sa valeur au milieu de l'exécution (arrêtée) du programme** (ainsi vous pouvez tester l'effet de continuer avec cette nouvelle valeur). **Arrêtez d'**abord l'exécution, puis **double-cliquez** sur la variable dont vous souhaitez suivre ou modifier la valeur. Pour surveiller simplement la valeur pendant l'exécution du programme, **laissez la boîte de dialogue ouverte**, puis l'une des commandes **Exécuter** ou

**Avancer d'un Pas** - sa valeur sera mise à jour dans **Modifier / Surveiller** selon les mêmes règles qui régissent les mises à jour dans le **Volet de Variables**. **Pour modifier la valeur de la variable**, remplissez la **valeur** de la boîte d'édition et **Acceptez**. Poursuivre l'exécution (en utilisant l'une des commandes **Avancer d'un Pas** ou **Exécuter**) pour utiliser cette nouvelle valeur à partir de ce point (ou vous pouvez revenir à la valeur précédente).

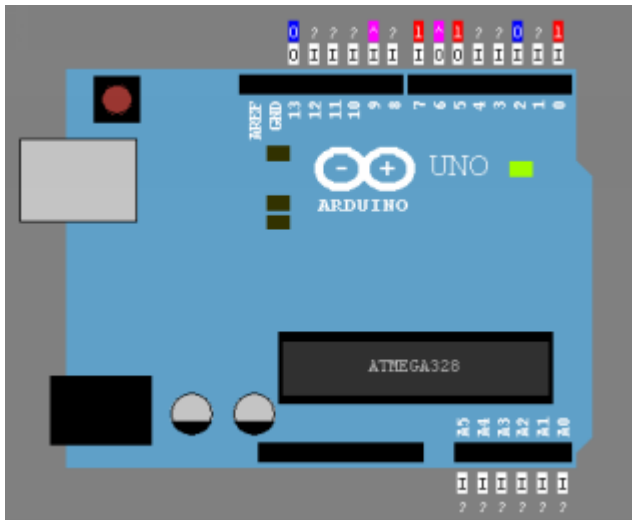
**Dans le programme Charger ou Réinitialiser**, notez que toutes les **variables de valeur non initialisées** sont réinitialisées à la valeur 0 et que toutes les **variables de pointeur non initialisées** sont réinitialisées à 0x0000.

## Volet du Banc de Laboratoire

La sous-fenêtre Banc de Laboratoire affiche une carte 'Uno' de 5 volts entourée d'un ensemble de dispositifs 'I / O' que vous pouvez sélectionner / personnaliser et vous connecter à vos broches 'Uno' souhaitées.

### Le 'Uno'

Ceci est une représentation de la carte 'Uno' et de ses LED embarquées. Lorsque vous chargez un nouveau programme dans UnoArduSim, s'il effectue une analyse réussie, il subit un "téléchargement simulé" vers le 'Uno' qui imite la façon dont se comporte réellement une carte 'Uno', les voyants série RX et TX clignotent (avec activité sur les broches 1 et 0 qui sont *câblées pour la communication série avec un ordinateur hôte* ). Ceci est immédiatement suivi d'un flash à 13 broches qui indique la réinitialisation de la carte et (et l'arrêt automatique de UnoArduSim à) le début de l'exécution du programme chargé. Vous pouvez éviter cet affichage et le décalage de chargement associé en désélectionnant **Afficher le Téléchargement** à partir de **Configurer | Préférences**.



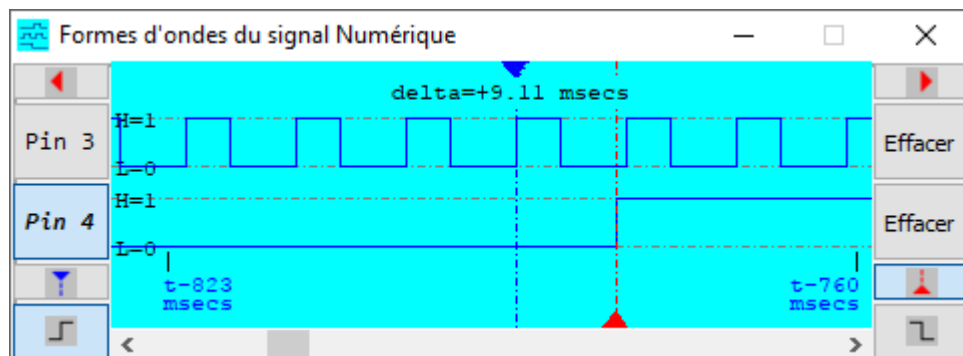
La fenêtre vous permet de visualiser les niveaux logiques numériques sur les 20 'Uno' broches ( '1' sur rouge pour 'HIGH' , '0' sur bleu pour 'LOW' , et '?' sur gris pour une tension indéterminée indéfinie), et les directions programmées ( 'I' pour 'INPUT' , ou 'O' pour 'OUTPUT' ). Pour les broches qui sont pulsées en utilisant PWM via 'analogWrite()' , ou par 'tone()' , ou par 'Servo.write()' , la couleur devient violet et le symbole affiché devient '^' .

Notez que **les broches numériques 0 et 1 sont câblées via des résistances de 1 kOhm à la puce USB pour la communication série avec un ordinateur hôte.**









A part: Les broches numériques 0-13 apparaissent comme broches du simulateur 0-13, et les broches



analogiques 0-5 apparaissent comme A0-A5. Pour accéder à une broche analogique dans votre programme, vous pouvez vous référer au numéro de broche par l'un des deux ensembles de nombres équivalents: 14-19; ou A0-A5 (A0-A5 sont des variables 'const' intégrées ayant des valeurs 14-19). Et seulement en utilisant 'analogRead()' , une troisième option est disponible - vous pouvez, pour cette instruction, supprimer le préfixe 'A' du numéro de broche et utiliser simplement 0-5. Pour accéder aux broches 14-19 de votre programme en utilisant 'digitalRead()' ou 'digitalWrite()' , vous pouvez simplement vous référer à ce numéro de broche, ou vous pouvez utiliser les alias A0-A5.

**Un clic gauche** sur n'importe quelle broche 'Uno' ouvre une fenêtre **Formes d'Onde Numériques** qui affiche la dernière **valeur d'une seconde** d'activité **numérique** sur cette broche. Vous pouvez cliquer sur les autres broches pour les ajouter à l'affichage des formes d'Onde Numériques (jusqu'à un maximum de 4 formes d'onde à la fois).



Cliquez sur la page à gauche ou à droite, ou utilisez les touches Accueil, PgUp, PgDn, Fin

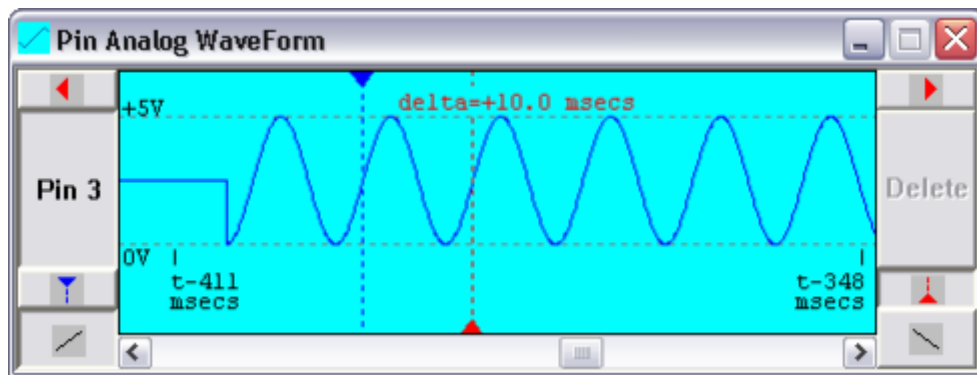
L'une des formes d'onde affichées sera la **forme d'onde de** broche active , indiquée par le fait que son bouton "Broche" est affiché comme étant enfoncé (comme dans la capture d'écran Formes d'Onde Numériques ci-dessus). Vous pouvez sélectionner une forme d'onde en cliquant sur le bouton Numéro de broche, puis sélectionner la polarité de bord d'intérêt en cliquant sur le bouton de sélection de la polarité du bord montant / descendant approprié, , ou  ou en utilisant les touches de raccourci flèches **haut** et **bas** . Vous pouvez ensuite **sauter** le curseur actif (soit les lignes de curseur bleu ou rouge avec leur temps delta indiqué) vers l'arrière ou vers l'avant jusqu'au bord numérique de la polarité choisie *de cette broche active* forme d' onde en utilisant les boutons du curseur ( ,  ou ,  selon le curseur activé précédemment avec  ou , ou utilisez simplement les touches du clavier ← et → .

Pour activer un curseur, cliquez sur son bouton d'activation coloré (  ou  montré ci-dessus) - *cela saute également - fait défiler la vue jusqu'à l'emplacement actuel de ce curseur* . Vous pouvez également alterner rapidement l'activation entre les curseurs (avec leurs vues respectivement centrées) à l'aide de la touche de raccourci **TAB** .







Vous pouvez **sauter** le curseur actuellement activé en **cliquant avec le bouton gauche n'importe où** dans la zone d'affichage de forme d'onde à l'écran. Vous pouvez également sélectionner la ligne de curseur rouge ou bleue en cliquant dessus à droite (pour l'activer), puis *faites-la glisser vers un nouvel emplacement* , et la libération. Lorsqu'un curseur souhaité est actuellement hors de l'écran, vous pouvez **cliquer avec le bouton droit de la souris n'importe où** dans la vue pour accéder au nouvel emplacement à l'écran. Si les deux curseurs sont déjà à l'écran, un clic droit suffit à alterner entre le curseur activé.

Pour ZOOMER et DÉZOOMER (le zoom est toujours centré sur le curseur ACTIVE), utilisez la molette de la souris ou les raccourcis clavier CTRL-flèche-vers-le-haut et CTRL-flèche-vers-le-bas .

Faire à la place un clic-droit sur n'importe quelle broche 'Uno' ouvre une fenêtre Forme d'Onde Analogique qui affiche la dernière valeur d' une seconde d' activité de niveau analogique sur cette broche. Contrairement à la fenêtre Formes d'Onde Numériques, vous pouvez afficher l'activité analogique sur une seule broche à la fois.



Cliquez sur la page à gauche ou à droite, ou utilisez les touches Accueil, PgUp, PgDn, Fin

Vous pouvez **sauter** les lignes de curseur bleu ou rouge au prochain "point de pente" montant ou descendant en utilisant les boutons fléchés vers l'avant ou vers l'arrière ( ,  ou ,  , toujours en fonction du curseur activé, ou utiliser les touches ← et → ) en même temps que les touches de sélection de pente ascendante / descendante ,  (le "point de pente" se produit lorsque la tension analogique passe par le seuil de haut niveau logique numérique de la broche ATmega). Vous pouvez également cliquer à nouveau pour sauter ou faire glisser ces lignes de curseur similaires à leur comportement dans la fenêtre Formes d'Onde Numériques

## Dispositifs 'I / O'

Un certain nombre de dispositifs différents entourent le 'Uno' sur le périmètre du **Volet du Banc de Laboratoire**. Les "petits" appareils 'I / O' (dont vous pouvez avoir jusqu'à 16 au total) résident le long des côtés gauche et droit du Volet. Les "grands" appareils 'I / O' (dont vous êtes autorisé jusqu'à 8 au total) ont des éléments "actifs" et se situent en haut et en bas du **Volet du Banc de Laboratoire**. Le nombre souhaité de chaque type de dispositif 'I / O' disponible peut être défini à l'aide du menu **Configurer | Dispositifs 'I / O'**.

Petits Dispositif 'I / O'		Grand Dispositifs d' 'I / O'	
Bouton-Poussoir	2	ServoMoteur	1
Commutateur Glissement	4	DC Moteur	1
Haut-parleur Piézo	2	Moteur Pas à Pas	1
DEL Colorée	6	Pulseur Numérique	1
Curseur Analogique	2	Générateur de Fonction	1
Total (max 16)	16	SFT 'Serial'	1
		I2C Esclave	1
		SPI Esclave	1
		SR Esclave	1
		Générateur Un-Tir	1
		Total (max 8)	8

Chaque dispositif 'I / O' a une ou plusieurs pièces jointes broche montré comme un **numéro de broche** à **deux chiffres** (00, 01, 02, ... 10, 11, 12, 13 et soit A0-A5, soit 14-19, après cela) dans une boîte d'édition correspondante. Pour les numéros de broche 2 à 9, vous pouvez simplement entrer le chiffre unique - le 0 en tête sera automatiquement fourni, mais pour les broches 0 et 1, vous devez d'abord entrer le 0 en tête. Les entrées sont *normalement* sur le côté gauche d'un dispositif 'I / O', et les sorties sont *normalement* sur la droite ( *si l'espace le permet* ). Tous les appareils 'I / O' répondront directement aux niveaux de broche et aux changements de niveau de broche, donc répondront aux fonctions de la bibliothèque ciblées sur leurs broches attachées, ou à '`digitalWrite()`' **programmé** (pour l'opération "bit-banged").

Vous pouvez connecter plusieurs dispositifs à la même broche ATmega à condition que cela ne crée pas de **conflit électrique**. Un tel conflit peut être créé soit par 'Uno' pin comme '**OUTPUT**' en conduisant avec un dispositif connecté à forte conduction (basse impédance) (par exemple, en pilotant une sortie 'FUNCGEN', ou une sortie 'MOTOR' encoder) ou par deux appareils connectés en concurrence (par exemple un 'PULSER' et un 'PUSH' - bouton attaché à la même broche). De tels conflits seraient désastreux dans une implémentation matérielle réelle et sont donc interdits, et seront signalés à l'utilisateur via une boîte de message pop-up).

La boîte de dialogue peut être utilisée pour permettre à l'utilisateur de choisir les types et les numéros des dispositifs 'I / O' souhaités. A partir de cette boîte de dialogue, vous pouvez également enregistrer les Dispositifs 'I / O' dans un fichier texte et / ou Charger les Dispositifs 'I / O' à partir d'un fichier texte préalablement enregistré (ou modifié) ( y compris toutes les connexions de broches et les paramètres cliquables). valeurs d'édition tapées dans la boîte ).

**Notez qu'à partir de la version 1.6, les valeurs dans les boîtes d'édition de période, de délai et de largeur d'impulsion dans les dispositifs IO concernés peuvent être suffixées avec la lettre 'S' (ou 's')**. Cela indique qu'ils doivent être mis à l' **échelle en** fonction de la position d'un curseur global '**I / O \_\_\_\_ S**' qui apparaît dans la **Barre d'Outils de la** fenêtre principale. Avec ce curseur complètement à droite, le facteur d'échelle est de 1,0 (unité), et avec le curseur complètement à gauche, le facteur d'échelle est de 0,0 (sous réserve des valeurs minimales imposées par chaque dispositif particulier 'I / O'). Vous pouvez mettre à l'échelle plusieurs valeurs de zone d'édition **simultanément à l'** aide de ce curseur. Cette fonction vous permet de faire glisser le curseur pendant l'exécution pour émuler facilement les largeurs d'impulsion, les périodes et les retards de modification pour les dispositifs 'I / O' joints.

Le reste de cette section fournit des descriptions pour chaque type d'appareil.

## 'Serial' Monitor ('SERIAL')

Ce dispositif 'I / O' permet l'entrée et la sortie série matérielle ATmega (via la puce 'Uno' USB) sur 'Uno' broches 0 et 1. Le débit en bauds est défini en utilisant la liste déroulante en bas - le débit en bauds sélectionné **doit correspondre à** la valeur que votre programme transmet à la fonction '`Serial.begin()`' pour une transmission / réception correcte. *La communication série est fixée à 8 bits de données, 1 bit d'arrêt et aucun bit de parité.*



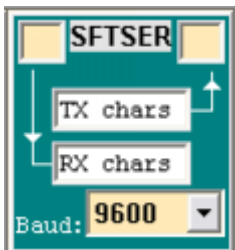
Pour envoyer une entrée au clavier à votre programme, tapez un ou plusieurs caractères dans la fenêtre d'édition supérieure (caractères TX), puis appuyez sur la touche **Retour**. (les caractères s'écrivent en italique pour indiquer que les transmissions ont commencé) - ou s'ils sont déjà en cours, les caractères ajoutés seront en italique. Vous pouvez ensuite utiliser les fonctions '`Serial.available()`' et '`Serial.read()`' pour lire les caractères dans l'ordre dans lequel ils ont été reçus dans le tampon 0 (le caractère tapé à gauche sera envoyé en premier). Les impressions textuelles et numériques formatées, ou les valeurs d'octets non formatées, peuvent être envoyées à la fenêtre de sortie de la console inférieure (caractères RX) en appelant les fonctions Arduino '`print()`',

'`println()`' ou '`write()`'.

En outre, une plus grande fenêtre de réglage / affichage des caractères TX et RX peut être ouverte en double-cliquant (ou un clic droit) sur ce 'SERIAL' dispositif. Cette nouvelle fenêtre a une plus grande zone d'édition des caractères TX, et un autre bouton 'Send' qui peut être cliqué pour envoyer les caractères TX au 'Uno' (sur la broche 0). Il y a aussi une option de case à cocher pour réinterpréter les séquences de caractères à échappement oblique inversé telles que '`\n`' ou '`\t`' pour l'affichage non-raw.

## Software Serial ('SFTSER')

Ce dispositif 'I / O' permet la gestion par le logiciel de bibliothèque, ou, alternativement, l'utilisateur "bit-banged", entrée série et sortie sur toute paire de 'Uno' broches que vous choisissiez de remplir (sauf pour les broches 0 et 1 sont dédiés au matériel 'Serial' communication). Votre programme doit avoir une ligne '`#include <SoftwareSerial.h>`' vers le haut si vous souhaitez utiliser les fonctionnalités de cette bibliothèque. Comme avec le matériel à 'SERIAL' appareil, la vitesse de transmission pour 'SFTSER' est réglé en utilisant la liste déroulante à sa base - la vitesse de transmission choisie doit correspondre à la valeur de votre programme passe à la '`begin()`' fonction pour bonne transmission / réception. *La communication série est fixée à 8 bits de données, 1 bit d'arrêt et aucun bit de parité.*



De même, comme avec le matériel '`Serial`', **une fenêtre plus grande pour le réglage / affichage TX et RX peut être ouverte en double-cliquant (ou en cliquant avec le bouton droit) sur le dispositif SFTSER.**

Notez que contrairement à l'implémentation matérielle de '`Serial`', aucune **mémoire** tampon TX ou RX fournie n'est prise en charge par les opérations internes d'interruption ATmega, de sorte que '`read()`' et '`write()`' bloquent (c'est-à-dire jusqu'à ce qu'ils soient terminés).



## Lecteur de Disque SD ('SD\_DRV')

Cet appareil 'I / O' permet des opérations d'entrée et de sortie à médiation logicielle (mais **pas** "bit-banged") sur les broches 'Uno' **SPI** (vous pouvez choisir la broche **CS\*** **que** vous utiliserez). Votre programme peut simplement `#include <SD.h>` ligne près du haut, et vous pouvez utiliser `<SD.h>` fonctions OU appeler directement `SdFile` fonctions vous-même.

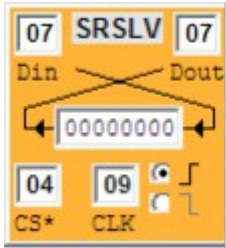


Une fenêtre plus grande affichant les répertoires et les fichiers (et le contenu) peut être ouverte en double-cliquant (ou en cliquant avec le bouton droit de la souris) sur le dispositif 'SD\_DRV'. Tout le contenu du disque est **chargé à partir** d'un sous-répertoire **SD** dans le répertoire de programme chargé (si elle existe) à `SdVolume init()`, **et se reflète à** ce même sous-répertoire dans le dossier `close()`, `remove()`, et sur `makeDir()` et `rmDir()`.

Une LED jaune clignote pendant les transferts SPI et 'DATA' indique le dernier octet de **réponse** 'SD\_DRV'. Tous les signaux SPI sont précis et peuvent être visualisés dans une **fenêtre de forme d'onde**.

## Esclave de Registre de Décalage ('SRSLV')

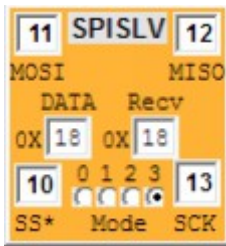
Cet appareil 'I / O' émule un simple dispositif de registre à décalage avec une broche **SS\*** ("slave-select") active-basse contrôlant la broche de sortie **Dout** (lorsque **SS\*** est haut, **Dout** n'est pas piloté). Votre programme peut utiliser les fonctionnalités de l'objet et de la bibliothèque SPI Arduino intégrés. Vous pouvez également choisir de créer vos propres signaux **Din** et **CLK** "bit-bang" pour piloter cet appareil.



Le dispositif détecte les transitions de bord sur son entrée **CLK** qui déclenchent le décalage de son registre - la polarité du bord **CLK** détecté peut être choisie en utilisant une commande par bouton radio. Sur chaque bord **CLK** (de la polarité détectée), le registre capture son niveau **Din** dans la position de bit le moins significatif (LSB) du registre à décalage, les bits restants étant simultanément décalés d'une position vers la position MSB. Lorsque **SS\*** est bas, la valeur actuelle dans la position MSB du registre à décalage est pilotée par **Dout**.

## Esclave SPI configurable ('SPISLV')

Cette 'I / O' dispositif de' émule un mode sélectionnable par l' esclave SPI avec une broche **SS\*** actif-bas ("slave-select") commandant la broche de sortie de **MISO** (lorsque **SS\*** est élevée, **MISO** est pas entraîné). Votre programme doit avoir une ligne `include <SPI.h>` si vous souhaitez utiliser la fonctionnalité de l'objet et de la bibliothèque SPI Arduino intégrés. Vous pouvez également choisir de créer vos propres signaux **MOSI** et **SCK** "bit-bang" pour piloter cet appareil.



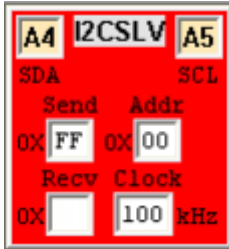
Le dispositif détecte les transitions de bord sur son entrée **CLK** en fonction du mode sélectionné ('MODE0', 'MODE1', 'MODE2' ou 'MODE3'), qui doit être choisi pour correspondre au mode SPI programmé de votre programme.

**En double-cliquant (ou en cliquant avec le bouton droit de la souris) sur l'appareil, vous pouvez ouvrir une fenêtre compagnon plus grande** qui vous permet de remplir un tampon de 32 octets maximum (afin d'émuler les dispositifs SPI qui renvoient automatiquement leurs données). voir les 32 derniers octets reçus (tous en tant que paires

hexadécimales). **Notez que** le prochain octet de tampon TX est automatiquement envoyé à 'DATA' seulement **après** que `SPI.transfer()` soit complet!

## Esclave I2C ('I2CSLV')

Cet appareil 'I / O' émule uniquement un *dispositif en mode esclave*. L'adresse du bus I2C peut être affectée à l'appareil à l'aide d'une entrée à deux chiffres dans sa zone d'édition 'Addr' (elle ne répond qu'aux transactions de bus I2C impliquant son adresse assignée). L'appareil envoie et reçoit des données sur sa broche **SDA** à vidange ouverte (pulldown uniquement) et répond au signal d'horloge de bus sur sa broche **SCL** à vidange ouverte (pulldown uniquement). Bien que 'Uno' soit le maître de bus responsable de générer le signal **SCL**, ce dispositif esclave réduira également **SCL** pendant sa phase basse afin d'étendre (si nécessaire) le temps de bus à un niveau approprié à sa vitesse interne (qui peut être défini dans sa '**Clock**' boîte d'éditer).



Votre programme doit avoir une ligne '#include <Wire.h>' si vous souhaitez utiliser la fonctionnalité de la bibliothèque 'TwoWire' pour interagir avec ce périphérique. Vous pouvez également choisir de créer vos propres données et signaux d'horloge bit-bang pour piloter cet appareil esclave.

Un seul octet pour la transmission vers 'Uno' master peut être défini dans la boîte d'édition 'Send', et un seul octet (le plus récemment reçu) peut être affiché dans son (lecture seule) 'Recv' edit-boîte. **Notez que** la valeur 'Send' boîte d'éditer reflète toujours l'octet suivant pour la transmission à partir du tampon de données interne de ce dispositif.

**En double-cliquant (ou en cliquant avec le bouton droit de la souris) sur l'appareil, vous pouvez ouvrir une fenêtre compagnon plus grande** qui vous permet de remplir un tampon FIFO de 32 octets maximum (de façon à émuler les dispositifs TWI dotés de telles fonctionnalités) et de visualiser (jusqu'à un maximum de 32) octets des dernières données reçues (sous la forme d'un affichage à deux chiffres hexadécimaux de 8 octets par ligne). Le nombre de lignes dans ces deux boîtes d'édition correspond à la taille du tampon TWI choisi (qui peut être sélectionné en utilisant **Configurer | Préférences**). Ceci a été ajouté en option puisque la bibliothèque Arduino '**wire.h**' utilise **cinq** tampons RAM de ce type dans son code d'implémentation, ce qui est coûteux en mémoire RAM. En éditant le **fichier** '**wire.h**' de l'installation Arduino pour modifier la constante '**BUFFER\_LENGTH**' (et en modifiant le fichier compagnon '**utility / twi.h**' pour changer la longueur du tampon TWI), les deux peuvent être 16 ou 8, un utilisateur *peut* Réduire de manière significative la surcharge de la mémoire RAM du 'Uno' dans une *implémentation matérielle* ciblée - UnoArduSim reflète donc cette possibilité réelle via **Configurer | Préférences**.

## Moteur Pas à Pas ('STEPR')

Cet appareil 'I / O' émule un moteur pas à pas bipolaire ou unipolaire 6V avec un contrôleur de pilote intégré piloté par **deux** signaux de commande (sur **P1**, **P2**) ou **quatre** (sur **P1**, **P2**, **P3**, **P4**). Le nombre de pas par tour peut également être réglé. Vous pouvez utiliser '**Stepper.h**' fonctions '**setSpeed()**' et '**step()**' pour piloter 'STEPR'. Sinon, 'STEPR' répondra *également* à vos propres **signaux de lecteur** '**digitalWrite()**' " bit-banged".



Le moteur est modélisé avec précision à la fois mécaniquement et électriquement. Les chutes de tension du moteur et les variations de réluctance et d'inductance sont modélisées avec un moment d'inertie réaliste par rapport au couple de maintien. L'enroulement du rotor du moteur a une résistance modélisée de  $R = 6$  ohms, et une inductance de  $L = 6$  milli-Henries qui crée une constante de temps électrique de 1,0 milliseconde. En raison de la modélisation réaliste, vous remarquerez que des impulsions de broche de commande très étroites *n'entraînent pas le déplacement* du moteur - à la fois en raison du temps de montée du courant fini et de l'effet de l'inertie du rotor. Ceci est en accord avec ce qui est observé lors de la conduite d'un vrai moteur pas à pas à partir d'un 'Uno' avec, bien sûr, une puce de pilote de moteur appropriée (**et nécessaire**) entre les fils du moteur et le 'Uno'!

Un bogue malheureux dans le code de la bibliothèque Arduino '**Stepper.h**' signifie qu'à la réinitialisation, le moteur pas à pas ne sera pas en position 1 (sur quatre étapes). Pour surmonter cela, l'utilisateur doit utiliser '**digitalWrite()**' dans sa routine '**setup()**' pour initialiser les niveaux de broche de contrôle aux niveaux



'**step** (1) ' appropriés à 2 broches (0,1) ou 4 -pin (1,0,1,0) contrôle, et permet au moteur 10 msec de se déplacer à la position initiale désirée du moteur 12-midi.

Notez que la **réduction d'engrenage n'est pas directement prise en charge** par manque d'espace, mais vous pouvez l'émuler dans votre programme en implémentant une variable compteur modulo-N et en appelant uniquement '**step** () ' lorsque ce compteur atteint 0 ).

### **Moteur DC ('MOTOR')**

Cet appareil 'I / O' émule un moteur DC à alimentation 100: 1 de 6 volts avec un contrôleur de pilotage intégré piloté par un signal de modulation de largeur d'impulsion (sur son entrée **Pwm** ) et un signal de contrôle de direction (sur son entrée **Dir** ). Le moteur dispose également d'une sortie de codeur de roue qui entraîne sa broche de sortie **Enc** . Vous pouvez utiliser '**analogWrite** () ' pour piloter la broche **Pwm** avec une fréquence d'onde PWM de 490 Hz (sur les broches 3,9,10,11) ou de 980 Hz (sur les broches 5,6) entre 0,0 et 1,0 ( '**analogWrite** () ' valeurs 0 à 255). Sinon, 'MOTOR' répondra *également* à vos propres signaux de lecteur '**digitalWrite** () ' " bit-banged".



Le moteur est modélisé avec précision à la fois mécaniquement et électriquement. La prise en compte des chutes de tension des transistors du moteur et du couple réaliste sans engrenage donne une vitesse maximale d'environ 2 tours par seconde et un couple de décrochage d'un peu plus de 5 kg-cm (avec un cycle de fonctionnement PWM constant de 1,0). total du moment d'inertie moteur-plus-charge de 2,5 kg-cm. L'enroulement du rotor du moteur a une résistance modélisée de  $R = 2$  ohms, et une inductance de  $L = 300$  micro-Henries qui crée une constante de temps électrique de 150 microsecondes. En raison de la modélisation réaliste, vous remarquerez que les impulsions PWM très étroites *n'obtiennent pas* le moteur de tourner - à la fois en raison du temps de montée du courant

fini, et le temps d'arrêt significatif après chaque impulsion étroite. Ceux-ci se combinent pour provoquer une impulsion de rotor insuffisante pour vaincre le retour de ressorts de type boîte de vitesses sous friction statique. La conséquence est que lorsque vous utilisez '**analogWrite** () ' , un cycle de fonctionnement inférieur à environ 0,125 n'entraînera pas le bougé du moteur - ceci est en accord avec ce qui est observé lorsque vous conduisez un véritable engrenage d'un 'Uno' avec, bien sûr, un ( **et nécessaire** ) module de pilote de moteur entre le moteur et le 'Uno'!

Le codeur moteur émulé est un capteur d'interruption optique monté sur l'arbre qui produit une forme d'onde de 50% avec 8 périodes de haut et bas par tour de roue (votre programme peut ainsi détecter les changements de rotation de la roue à 22,5 degrés).

### **ServoMoteur ('SERVO')**

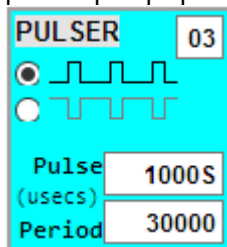
Ce dispositif 'I / O' émule un servomoteur DC d'alimentation 6 V commandé par PWM et commandé en position. Les paramètres de modélisation mécanique et électrique pour le fonctionnement du servo correspondent étroitement à ceux d'un servo HS-422 standard. Le servo a une vitesse de rotation maximale d'environ 60 degrés en 180 ms . Si la case à cocher en bas à gauche est cochée, le servo devient un servo de rotation continue avec la même vitesse maximale, mais maintenant la largeur de pulsation PWM définit la vitesse plutôt que l'angle



Votre programme doit avoir une ligne '**include <servo.h>**' avant de déclarer votre (vos) instance (s) Servo si vous choisissez d'utiliser la fonctionnalité de la bibliothèque Servo , par exemple '**Servo.write** ()', '**Servo.writeMicroseconds** () ' Alternativement, SERVO répond également aux signaux '**digitalWrite** () ' " bit-banged". En raison de l'implémentation interne de UnoArduSim, vous êtes limité à 5 'SERVO' dispositifs.

## Pulseur Binaire ('PULSER')

Ce dispositif 'I / O' émule un générateur de forme d'onde à impulsion numérique simple qui produit un signal périodique qui peut être appliqué à n'importe quelle broche 'Uno' choisie.

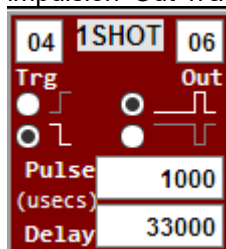


Les périodes et les largeurs d'impulsion (en microsecondes) peuvent être définies à l'aide de boîtes d'édition - la période minimale autorisée est de 50 microsecondes et la largeur d'impulsion minimale est de 10 microsecondes. La polarité peut également être choisie: impulsions frontales positives (0 à 5V) ou impulsions négatives (5V à 0V).

Les valeurs 'Pulse' et 'Period' sont modulables à partir du curseur principal **de la Barre d'Outils** 'I / O \_\_\_\_\_ S' en ajoutant un suffixe (ou les deux) à la lettre 'S' (ou 's').

## Un-Tir ('1SHOT')

Cet appareil 'I / O' émule un Un-Tir numérique qui peut générer une impulsion de polarité et de largeur d'impulsion choisie sur son 'Out' broche, survenant après un délai spécifié à partir d'un front de déclenchement reçu sur sa broche d'entrée **Trg** (trigger). Une fois que le front de déclenchement spécifié est reçu, le chronométrage commence et une nouvelle impulsion de déclenchement ne sera pas reconnue tant que l'impulsion 'Out' n'aura pas été produite (et si elle est complètement terminée).

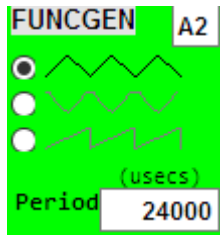


Une utilisation possible de ce dispositif est de simuler des capteurs de mesure d'échographie qui génèrent une impulsion de distance en réponse à une impulsion de déclenchement. Il peut également être utilisé partout où vous souhaitez générer un signal d'entrée de broche synchronisé (après le délai choisi) sur un signal de sortie de broche créé par votre programme.

**Les valeurs 'Pulse' et 'Delay' sont modulables à partir du curseur de la fenêtre principale Barre d'Outils 'I / O \_\_\_\_\_ S' en ajoutant un suffixe (ou les deux) à la lettre 'S' (ou 's').**

## Générateur de Fonction Analogique ('FUNCGEN')

Ce dispositif 'I / O' émule un générateur de forme d'onde analogique simple qui produit un signal périodique qui peut être appliqué à n'importe quelle broche 'Uno' choisie.



La période (en microsecondes) peut être définie à l'aide de la zone d'édition - la période minimale autorisée est de 100 microsecondes. La forme d'onde créée peut être sinusoïdale, triangulaire ou en dents de scie (pour créer une onde carrée, utilisez plutôt 'PULSER'). À des périodes plus courtes, moins d'échantillons par cycle sont utilisés pour modéliser la forme d'onde produite (seulement 4 échantillons par cycle à période = 100 usecs).

La valeur 'Period' est extensible à partir du curseur de la fenêtre principale **Barre d'Outils** 'I / O \_\_\_\_\_ S', en lui attribuant un suffixe avec la lettre 'S' (ou 's').

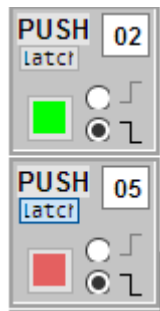
## Haut-parleur Piézoélectrique ('PIEZO')



Ce dispositif vous permet d'écouter les signaux sur n'importe quelle broche 'Uno' et peut être un complément utile aux DEL pour le débogage de votre programme. Vous pouvez aussi vous amuser à jouer des sonneries par `'tone()'` et `'delay()'` calls (bien qu'il n'y ait pas de filtrage de la forme d'onde rectangulaire, donc vous n'entendrez pas de notes "pures").

Vous pouvez également écouter un appareil 'PULSER' ou 'FUNCEN' connecté en accrochant un 'PIEZO' à la broche que l'appareil pilote.

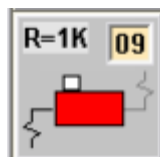
## Bouton Poussoir ('PUSH')



Ce dispositif d'I/O émule un **bouton-poussoir unipolaire, unipolaire (SPST) à enclenchement momentané**, normalement ouvert, **avec une résistance de 10 k ohms (ou pull-down)**. Si une sélection de transition de front montant est choisie pour l'appareil, les contacts du bouton-poussoir seront câblés entre la broche de l'appareil et + 5V, avec une mise à la terre de 10 k-Ohm. Si une transition de front descendant est choisie pour l'appareil, les contacts du bouton-poussoir seront câblés entre la broche de l'appareil et la masse, avec un pull-up de 10 kΩ à + 5V.

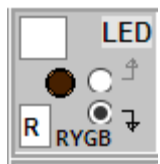
En faisant un clic gauche sur le bouton, ou appuyez sur n'importe quelle touche, vous fermez le contact du bouton-poussoir. En mode **momentané**, il reste fermé tant que vous maintenez le bouton de la souris, ou la touche, enfoncé, et en mode **verrou** (activé en cliquant sur le bouton @ xxxlatch @) il reste fermé (et de couleur différente) jusqu'à ce que vous cliquiez à nouveau sur le bouton, ou appuyez à nouveau une touche.

## Résistance de l'interrupteur à Glissière ('R = 1K')



Ce dispositif permet à l'utilisateur de se connecter à une broche 'Uno' soit une résistance pull-up de 1 k-Ohm à + 5V, soit une résistance pull-down de 1 k-ohm à la masse. Cela vous permet de simuler les charges électriques ajoutées à un dispositif matériel réel. En cliquant avec le bouton gauche de la souris sur le **corps de l'interrupteur à glissière**, vous pouvez basculer vers la sélection souhaitée. L'utilisation d'un ou de plusieurs de ces dispositifs vous permettrait de définir un "code" unique (ou multi) pour votre programme à lire et à répondre.

## DEL Colorée ('LED')



Vous pouvez connecter une DEL entre la broche choisie 'Uno' (à travers une résistance de limitation de courant intégrée cachée de la série 1 k-Ohm) à la terre ou à + 5V - cela vous donne le choix de faire allumer la DEL lorsque le connecté 'Uno' pin est **'HIGH'**, ou à la place, quand il est **'LOW'**.

La couleur de la DEL peut être choisie soit en rouge ('R'), en jaune ('Y'), en vert ('G') ou en bleu ('B') en utilisant sa boîte d'édition.





## Curseur Analogique








Un potentiomètre 0-5V commandé par curseur peut être connecté à n'importe quelle broche 'Uno' choisie pour produire un niveau de tension analogique statique (ou changeant lentement) qui serait lu par `'analogRead()'` comme une valeur de 0 à 1023. Utilisez la souris pour faire glisser, ou cliquez pour sauter, le curseur analogique.

## Menus









### Fichier:

 <b>Charger INO ou PDE Prog ( ctrl-L )</b>	Permet à l'utilisateur de choisir un fichier programme ayant l'extension sélectionnée. Le programme est immédiatement analysé
<b>Éditer / Examiner (ctrl-E )</b>	Ouvre le programme chargé pour l'affichage / l'édition.
 <b>Enregistrer</b>	Enregistrez le contenu du programme édité dans le fichier programme d'origine.
<b>Enregistrer Sous</b>	Enregistrez le contenu du programme édité sous un nom de fichier différent.
 <b>Suivant ( '#inclure' )</b>	Avance le Volet de Code pour afficher le prochain fichier '#include'
 <b>Précédent</b>	Renvoie l' affichage du <b>Volet de Code</b> dans le fichier précédent
<b>Quitter</b>	Quitte UnoArduSim après avoir rappelé à l'utilisateur d'enregistrer les fichiers modifiés.

### Trouver:

 <b>Trouver la Fonction/Var Suivante</b>	Accédez à la fonction suivante dans le <b>Volet de Code</b> (s'il a le focus actif), ou à la variable suivante dans le <b>Volet de Variables</b> (si à la place il a le focus actif).
 <b>Trouver la Fonction/Var Précédenter</b>	Accédez à la fonction précédente dans le <b>Volet de Code</b> (s'il a le focus actif), ou à la variable précédente dans le <b>Volet de Variables</b> (si à la place, il a le focus actif).
 <b>Définir le Texte de Recherche (ctrl-F)</b>	Activez la case d'édition Rechercher dans la <b>Barre d'Outils</b> pour définir le texte à rechercher ensuite (et ajoutez le premier mot de la ligne en surbrillance dans le <b>Volet de Code</b> ou dans le <b>Volet de Variables</b> si l'un d'eux a le focus).
 <b>Trouver le Texte suivant</b>	Accédez à l'occurrence de texte suivante dans le <b>Volet de Code</b> (s'il a le focus actif), ou à l'occurrence de texte suivante dans le <b>Volet de Variables</b> (si à la place il a le focus actif).
 <b>Trouver le Texte précédent</b>	Accédez à l'occurrence de texte précédente dans le <b>Volet de Code</b> (s'il a le focus actif), ou à l'occurrence de texte précédente dans le <b>Volet de Variables</b> (si à la place il a le focus actif).

## Exécuter:

 <b>Un Pas Dans (F4)</b>	Avancez l'exécution en avant par une instruction, ou <i>dans une fonction appelée</i> .
 <b>Un Pas Franchir (F5)</b>	Avancez l'exécution vers l'avant par une instruction ou <i>par un appel de fonction complet</i> .
 <b>Un Pas Sortir (F6)</b>	Avance l'exécution de <i>juste assez pour quitter la fonction en cours</i> .
 <b>Exécuter Vers (F7)</b>	Exécute le programme, en <i>s'arrêtant à la ligne de programme souhaitée</i> - vous devez d'abord cliquer pour mettre en surbrillance une ligne de programme souhaitée avant d'utiliser Exécuter-Vers.
 <b>Exécuter Jusqu'à (F8)</b>	Exécute le programme jusqu'à ce qu'une écriture se produise dans la variable ayant la surbrillance en cours dans le <b>Volet de Variables</b> (cliquez sur l'un pour établir la surbrillance initiale).
 <b>Exécuter (F9)</b>	Exécute le programme.
 <b>Arrêter (F10)</b>	Arrête l'exécution du programme ( <i>et gèle le temps</i> ).
 <b>Réinitialisez</b>	Réinitialise le programme (toutes les variables de valeur sont réinitialisées à la valeur 0 et toutes les variables de pointeur sont réinitialisées sur 0x0000).
<b>Animer</b>	Intervient automatiquement les lignes de programme consécutives avec <i>un retard artificiel ajouté</i> et la mise en surbrillance de la ligne de code actuelle. L'opération en temps réel et les sons sont perdus.
<b>Ralenti</b>	Ralentit le temps d'un facteur de 10.

## Options:

<b><u>Enjamber des Structors / Opérateurs</u></b>	Voler à travers les constructeurs, les destructeurs, et les fonctions de surcharge de l'opérateur pendant tout pas (c'est-à-dire qu'il ne s'arrêtera pas à l'intérieur de ces fonctions).
<b><u>Modélisation de Registre-Allocation</u></b>	Affectez des locaux de fonction pour libérer les registres ATmega au lieu de les affecter à la pile (génère une utilisation de RAM quelque peu réduite).
<b><u>Erreur sur non-Initialisé</u></b>	Signaler comme une erreur d'analyse n'importe où votre programme tente d'utiliser une variable sans avoir initialisé sa valeur (ou au moins une valeur dans un tableau).
<b><u>'loop()' Delay ajouté</u></b>	Ajoute 200 microsecondes de retard à chaque fois que 'loop()' est appelé (dans le cas où aucun autre appel de programme à 'delay()' anywhere) - utile pour éviter de tomber trop loin derrière le temps réel.

## Configurer:

<b><u>Dispositifs 'I / O'</u></b>	Ouvre une boîte de dialogue pour permettre à l'utilisateur de choisir le (s) type (s) et les numéros des dispositifs 'I / O' souhaités. A partir de cette boîte de dialogue, vous pouvez également enregistrer les Dispositifs 'I / O' dans un fichier texte et / ou charger les Dispositifs 'I / O' à partir d'un fichier texte préalablement enregistré (ou modifié) (y compris tous les connexions de pin et les paramètres cliquables). valeurs
<b><u>Préférences</u></b>	Ouvre une boîte de dialogue pour permettre à l'utilisateur de définir les préférences, y compris la syntaxe Expert, le choix de la police de caractère, opter pour une taille de police plus grande, appliquer les limites des tableaux, autoriser les mots-clés logiques, afficher le téléchargement de la version de carte 'Uno' et de la longueur du tampon TWI (pour les dispositifs I2C).

## VarRafraichir :

<b><u>Autoriser la Contraction automatique (-)</u></b>	Autoriser UnoArduSim à réduire les tableaux / objets développés affichés en cas de retard sur le temps réel.
<b><u>Autoriser la Réduction</u></b>	Autoriser une fréquence réduite des mises à jour de l'affichage dans le <b>Volet de Variables</b> pour éviter le scintillement ou réduire la charge du processeur en cas de retard - les valeurs affichées ne sont mises à jour que périodiquement, <b><i>mais aussi chaque fois que le programme est arrêté</i></b> .
<b><u>Minimal</u></b>	Actualisez l' affichage du <b>Volet de Variables seulement</b> 4 fois par seconde.
<b><u>Modifications de HighLight</u></b>	Mettez en surbrillance la dernière variable modifiée (peut provoquer un ralentissement).

## Fenêtres:

<b><u>'Serial' Monitor</u></b>	Connectez un dispositif 'I/O' série aux broches 0 et 1 (s'il n'y en a pas) et augmentez la fenêtre de texte TX / RX 'Serial' monitor.
<b><u>Tout restaurer</u></b>	Restaurez toutes les fenêtres enfants réduites.
<b><u>Formes d'Onde Numériques</u></b>	Restaurez une fenêtre Formes d'Onde Numériques minimisée.
<b><u>Forme d'Onde Analogique</u></b>	Restaurez une fenêtre Forme d'onde analogique réduite.

## Aidez-moi:

<b><u>Fichier d'aide rapide</u></b>	Ouvre le fichier PDF UnoArduSim_QuickHelp.
<b><u>Fichier d'aide complet</u></b>	Ouvre le fichier PDF UnoArduSim_FullHelp.
<b><u>Corrections de bugs</u></b>	Voir les corrections de bugs importants depuis la version précédente.
<b><u>Changement / Améliorations</u></b>	Voir les changements importants et les améliorations depuis la version précédente.
<b><u>Sur</u></b>	Affiche la version, copyright.

## La Modélisation

### Carte 'Uno' et Les Dispositifs 'I / O'

Les appareils 'Uno' et 'I / O' attachés sont tous modélisés électriquement avec précision, et vous serez en mesure d'avoir une bonne idée de la façon dont vos programmes se comportent avec le matériel réel, et tous les conflits de broches électriques seront signalés.

### Synchronisation

UnoArduSim s'exécute assez rapidement sur un PC ou une tablette pour pouvoir ( *dans la majorité des cas* ) modéliser les actions du programme en temps réel, **mais seulement si votre programme incorpore** au moins quelques petits appels `'delay()'` ou d'autres appels qui resteront naturellement il est synchronisé en temps réel (voir ci-dessous).

Pour ce faire, UnoArduSim utilise une fonction de temporisation de rappel Windows, ce qui lui permet de garder une trace précise du temps réel. L'exécution d'un certain nombre d'instructions de programme est simulée pendant une tranche de temporisateur et les instructions qui nécessitent une exécution plus longue (comme les appels à `'delay()'` ) peuvent nécessiter l'utilisation de plusieurs tranches de temporisation. Chaque itération de la fonction de temporisation de rappel corrige l'heure du système en utilisant l'horloge matérielle du système, de sorte que l'exécution du programme soit constamment ajustée pour rester en verrouillage en temps réel. **Les seules fois où le taux d'exécution doit être inférieur au temps réel, c'est lorsque l'utilisateur a créé des boucles serrées** sans retard supplémentaire , ou que les dispositifs 'I / O' sont configurés pour fonctionner avec des fréquences 'I / O' très élevées (et / ou baud rate) qui générerait un nombre excessif d'événements de changement de niveau de broche et de surcharge de traitement associée. UnoArduSim fait face à cette surcharge en sautant certains intervalles de minuterie pour compenser, ce qui ralentit la progression du programme en **dessous du temps réel** .

De plus, les programmes avec de grandes baies affichées, ou encore avec des boucles serrées **sans délai supplémentaire** peuvent provoquer une fréquence d'appel haute et générer une mise à jour élevée de l'affichage du **Volet de Variables en retardant le** temps réel - cela peut être contourné en utilisant **Autoriser Réduction** , ou en cas de besoin, **Minimal**, à partir du menu **VarRafraichir** ,

La modélisation précise du temps d'exécution inférieur à la milliseconde pour chaque instruction ou opération de programme **n'est pas effectuée** - seules des estimations très approximatives pour la plupart ont été adoptées à des fins de simulation. Cependant, le timing des fonctions `'delay()'` et `'delayMicroseconds()'` et des fonctions `'millis()'` et `'micros()'` est parfaitement exact, et **tant que vous utilisez au moins une des fonctions de délai** dans une boucle quelque part dans votre programme, **ou** vous utilisez une fonction qui se lie naturellement à une opération en temps réel (comme `'print()'` qui est liée à la vitesse de transmission choisie), alors la performance simulée de votre programme sera très proche en temps réel (encore une fois, en interdisant des événements de changement de niveau de broche haute fréquence excessivement excessifs ou des mises à jour excessives de variables autorisées par l'utilisateur qui pourraient le ralentir).

Afin de voir l'effet des instructions individuelles du programme en cours d' *exécution* , il peut être souhaitable de pouvoir ralentir les choses. Un facteur de ralentissement temporel de 10 peut être défini par l'utilisateur dans le menu **Exécuter** .

### Dispositifs 'I / O'

Ces dispositifs virtuels reçoivent une signalisation en temps réel des changements qui se produisent sur leurs broches d'entrée, et produisent des sorties correspondantes sur leurs broches de sortie qui peuvent ensuite être détectées par le 'Uno' - elles sont donc intrinsèquement synchronisées à l'exécution du programme. La temporisation interne d'un dispositif 'I / O' est définie par l'utilisateur (par exemple, via la sélection du débit en bauds ou la fréquence de l'horloge), et les événements du simulateur sont configurés pour suivre le fonctionnement interne en temps réel.d'un



## Des Sons

Chaque appareil 'PIEZO' produit un son correspondant aux changements de niveau électrique sur la broche attachée, quelle que soit la source de ces changements. Pour que les sons soient synchronisés avec l'exécution du programme, UnoArduSim démarre et arrête la lecture d'un tampon sonore associé lorsque l'exécution est démarrée / arrêtée.

A partir de V2.0, le son a été modifié pour utiliser l'API audio Qt - malheureusement son QAudioOutput 'class' ne supporte pas le buffer audio pour éviter de manquer d'échantillons sonores (comme cela peut arriver pendant les longs délais d'exploitation de la fenêtre OS). Par conséquent, afin d'éviter la grande majorité des clics sonores agaçants et la rupture du son pendant les retards de l'OS, le son est maintenant coupé selon la règle suivante:

Le son est coupé tant que UnoArduSim n'est pas la fenêtre "active" (sauf lorsqu'une nouvelle fenêtre enfant vient d'être créée et activée), **et même** si UnoArduSim est la fenêtre principale "active" mais que le pointeur de la souris est en **dehors** de sa fenêtre principale. zone de la fenêtre client.

Notez que ceci implique que le son sera temporairement coupé comme le pointeur de la souris **survole une fenêtre enfant**, et sera mis en sourdine **si cette fenêtre enfant est cliquée pour l'activer** (jusqu'à ce que la fenêtre UnoArduSim nain soit de nouveau cliquée pour réactiver ) .

***Le son peut toujours être désactivé en cliquant sur n'importe où dans la zone cliente de la fenêtre principale UnoArduSim.***

En raison de la mise en mémoire tampon, le système dispose d'un décalage en temps réel allant jusqu'à 250 millisecondes par rapport à l'événement correspondant sur la broche du 'PIEZO' ci-joint.

## Limitations et éléments non pris en charge

### Fichiers Inclus

A '<>' - entre crochets '#inclure' de '<Servo.h>', '<Wire.h>', '<SoftwareSerial.h>', '<SPI.h>', '<EEPROM.h>' et '<SD.h>' est supporté, mais ceux-ci sont seulement émulés - les fichiers réels ne sont pas recherchés; Au lieu de cela, leur fonctionnalité est directement "intégrée" dans UnoArduSim, et est valable pour la version Arduino supportée fixe.

Tout' cité 'include' (par exemple "supp.ino", "myutil.cpp" ou "mylib.h") est pris en charge, mais tous ces fichiers doivent **résider dans le même répertoire comme le fichier de programme parent** contient leur '#include' (il n'y a pas de recherche effectuée dans d'autres répertoires). La fonction '#include' peut être utile pour minimiser la quantité de code de programme affichée dans le **Volet de Code** à un moment donné. Les fichiers d'en-tête avec '#include' (ceux ayant une extension ".h") provoqueront également la tentative du simulateur incluant le fichier du même nom ayant une extension ".cpp" (s'il existe aussi dans le répertoire du parent programme).

### Allocation de mémoire dynamique et RAM

Les opérateurs 'new' et 'delete' sont pris en charge, tout comme les objets natifs Arduino 'String', **mais pas les appels directs vers** 'malloc()', 'realloc()' et 'free()' que ceux-ci s'appuient sur.

L'utilisation excessive de RAM pour les déclarations de variables est signalée au moment de l'analyse, et le débordement de la mémoire RAM est signalé pendant l'exécution du programme. Un élément sur menu **Options** vous permet d'émuler l'allocation de registre ATmega normale comme le ferait le compilateur AVR, ou de modéliser un autre schéma de compilation qui utilise uniquement la pile (comme option de sécurité au cas où un

bug apparaîtrait dans mon allocation de registre la modélisation). Si vous deviez utiliser un pointeur pour examiner le contenu de la pile, il devrait refléter exactement ce qui apparaîtrait dans une implémentation matérielle réelle.

## Allocations de mémoire 'Flash'

'Flash' memory 'byte', 'int' et 'float' variables / tableaux et leurs fonctions d'accès en lecture correspondantes sont supportées. Tout 'F()' L'appel de fonction ("Flash'-macro") de n'importe quelle chaîne littérale **est** supporté, mais les seules fonctions d'accès direct 'Flash' -memory **supportées** sont 'strcpy\_P()' et 'memcpy\_P()', donc d'utiliser d'autres fonctions vous devrez d'abord copier la chaîne 'Flash' dans une variable RAM 'String' normale, puis travailler avec cette RAM 'String'. Lorsque vous utilisez le mot-clé 'PROGMEM' variable-modificateur, il doit apparaître **devant** le nom **de** la variable et cette variable **doit également être déclarée** comme 'const'.

## 'String' Variables

La bibliothèque native 'String' est presque entièrement supportée avec quelques exceptions très (et mineures).

Les 'String' opérateurs pris en charge sont +, + =, <, <=, >, > =, ==, !=, et []. Notez que: 'concat()' prend un **seul** argument qui est 'String', ou 'char', ou 'int' à **ajouter** à l'objet original 'String', **pas** deux arguments comme cela est indiqué par erreur sur la référence Arduino les pages Web).

## Bibliothèques Arduino

Seulement 'Stepper.h', 'SD.h', 'Servo.h', 'SoftwareSerial.h', 'SPI.h', 'Wire.h' et 'EEPROM.h' pour la version **Arduino V1.6.6** sont actuellement pris en charge dans UnoArduSim. Essayer '#include' les fichiers ".cpp" et ".h" d'autres bibliothèques non encore supportées **ne fonctionneront pas** car elles contiendront des instructions d'assemblage de bas niveau et des directives non supportées et des fichiers non reconnus!

## Pointeurs

Les pointeurs vers des types simples, des tableaux ou des objets sont tous pris en charge. Un pointeur peut être égal à un tableau du même type (par exemple `iptr = intarray`), mais il n'y aurait **pas de vérification ultérieure des limites de tableaux** sur une expression comme `iptr [index]`.

Les fonctions peuvent renvoyer des pointeurs, ou 'const' pointeurs, mais tout niveau suivant de 'const' sur le pointeur retourné est ignoré.

Les appels de fonction **ne** sont **pas pris en charge** par les **pointeurs de fonction déclarés** par l'**utilisateur**.

## Objets 'class' et 'struct'

Bien que le poly-morphisme et l'héritage (à n'importe quelle profondeur) soient supportés, un 'class' ou 'struct' ne peut être défini que pour avoir au plus **une** base 'class' (ie l'héritage **multiple** n'est pas supporté). Les appels d'initialisation de base 'class' constructor (via la notation deux-points) dans les lignes de déclaration du constructeur sont pris en charge, mais **pas** les initialisations de membre utilisant la même notation deux-points. Cela signifie que les objets qui contiennent des **variables** 'const' non 'static', ou des variables de type référence, ne sont pas supportés (ceux-ci ne sont possibles qu'avec des initialisations de membres à l'instant de construction).

Les surcharges d'opérateur de copie sont prises en charge avec les constructeurs de mouvement et les affectations de déplacement, mais les fonctions de conversion d'objet définies par l'utilisateur ("cast-like") ne sont pas prises en charge.

## Portée

Il n'y a pas de support pour le mot clé **'using'** , ou pour les espaces de noms, ou pour **'file'** scope. Toutes les déclarations non locales sont supposées être globales.

Tout **'typedef'** , **'struct'** , ou **'class'** définition (c'est-à-dire qui peut être utilisé pour de futures déclarations), doit être rendu **global** ( *les* définitions **locales** de tels éléments dans une fonction ne sont pas supportées).

## Qualificateurs **'unsigned'** , **'const'** , **'volatile'** , **'static'**

Le préfixe **'unsigned'** fonctionne dans tous les contextes juridiques normaux. Le mot clé **'const'** , lorsqu'il est utilisé, doit **précéder** le nom de la variable ou le nom de la fonction ou **'typedef'** nom qui est déclaré - le placer après le nom provoquera une erreur d'analyse. Pour les déclarations de fonction, seules les fonctions renvoyant le pointeur peuvent avoir **'const'** dans leur déclaration.

Toutes les variables UnoArduSim sont **'volatile'** par implémentation, donc le mot clé **'volatile'** est simplement ignoré dans toutes les déclarations de variables. Les fonctions ne sont pas autorisées à être déclarées **'volatile'** , ni les arguments d'appel de fonction.

Le mot-clé **'static'** est autorisé pour les variables normales et pour les membres-objets et les fonctions-membres, mais explicitement interdit pour les instances d'objet elles-mêmes ( **'class'** / **'struct'** ), pour les fonctions non-membres.

## Directives du Compilateur

**'#include'** et regular **'#define'** sont tous deux supportés, mais ***pas de macro*** **'#define'** . Le **'#pragma'** directives d'inclusion directive et conditionnelle ( **'#ifdef'** , **'#ifndef'** , **'#si'** , **'#endif'** , **'#else'** et **'#elif'** ) ne sont ***pas non plus supportés*** . La ligne **'#'** , **'#erreur'** et les macros prédéfinies (comme **'\_LINE\_'** , **'\_FILE\_'** , **'\_DATE\_'** et **'\_TIME\_'** ) ne sont ***pas non plus supportées***.

## Éléments de Langage Arduino

Tous les éléments du langage Arduino natif sont pris en charge à l'exception de l' instruction **'goto'** douteuse (la seule utilisation raisonnable pour ce que je peux penser serait comme un saut (à un bouclage et une boucle sans fin d'arrêt de sécurité) dans le cas de une condition d'erreur que votre programme ne peut pas traiter autrement).

## C / C ++ - Éléments de Langage

Les "qualificateurs de champ de bits" qui économisent du bit pour les membres dans les définitions de structure **ne** sont ***pas pris en charge*** .

**'union'** n'est ***pas supporté***.

L'opérateur de virgule "oddball" n'est ***pas supporté*** (donc vous ne pouvez pas exécuter plusieurs expressions séparées par des virgules quand une seule expression est normalement attendue, par exemple dans **'while()'** et **'for (;)'** constructions).

## Modèles de Fonction

Les fonctions définies par l'utilisateur qui utilisent le mot-clé "template" pour lui permettre d'accepter les arguments de type "generic" **ne** sont ***pas supportées*** .

## Émulation en Temps Réel

Comme indiqué ci-dessus, les temps d'exécution des différentes instructions du programme Arduino **ne** sont ***pas*** modélisés avec précision, de sorte que pour fonctionner en temps réel, votre programme aura besoin d'une sorte d' **instruction** `'delay()'` dominante (au moins une fois par `'loop()'` ), ou une instruction qui est naturellement synchronisée avec les changements au niveau de la broche en temps réel (tels que `'pulseIn()'` , `'shiftIn()'` , `'Serial.read()'` , `'Serial. print()'` , `'Serial.flush()'` etc. ).

Voir **Modélisation** et **sons** ci-dessus pour plus de détails sur les limitations.

## **Notes de Version**

### **Corrections de Bugs**

#### **V2.0.1- Janv. 2018**

- 1) Dans les langues autres que l'anglais, '**en**' a été incorrectement affiché comme étant sélectionné dans les **Préférences**, ce qui rend le retour à l'anglais difficile (nécessitant une désélection puis une nouvelle sélection).
- 2) L'utilisateur a pu laisser une valeur de boîte d'édition de broche d'appareil dans un état incomplet (comme '**A\_**' ) et laisser les bits 'DATA' d'un 'SRSlave' incomplets.
- 3) Le nombre maximum de Curseurs Analogiques a été limité à 4 (corrigé maintenant à 6).
- 4) UnoArduSim n'insiste plus sur l'apparition de '=' dans une initialisation d'agrégat de tableau.
- 5) UnoArduSim avait insisté pour que l'argument 'inverted\_logic' soit fourni à '**SoftwareSerial()**'
- 6) Les opérations de décalage de bits permettent maintenant des décalages plus longs que la taille de la variable décalée.

#### **V2.0- Déc. 2017**

- 1) Toutes les fonctions qui ont été déclarées comme '**unsigned**' avait néanmoins renvoyé des valeurs comme si elles étaient '**signed**'. Cela n'a eu aucun effet si la valeur '**return**' a été affectée à '**unsigned**' variable, mais aurait causé une mauvaise interprétation négative si elle avait MSB == 1, et elle a ensuite été affectée à une variable '**signed**', ou testée dans une inégalité.
- 2) Les curseurs analogiques n'atteignaient que la valeur maximale '**analogRead()**' de 1022, pas la valeur 1023 correcte.
- 3) Un bogue introduit par inadvertance dans V1.7.0 dans la logique utilisée pour accélérer le traitement de la broche SCK du système SPI a provoqué l'échec des transferts SPI pour '**SPI\_MODE1**' et '**SPI\_MODE3**' après le premier octet transféré (une transition parasite SCK supplémentaire suivait chaque octet). Les mises à jour d'une boîte 'SPISLV' edit 'DATA' pour les octets transférés ont également été retardées,
- 4) L'appareil à DEL de couleur ne mentionnait pas 'B' (pour Bleu) comme option de couleur (même si elle était acceptée).
- 5) Les paramètres des dispositifs 'SPISLV' et 'I2CSLV' n'étaient pas enregistrés dans le fichier de l'utilisateur des **Dispositifs 'I / O'**.
- 6) La copie des instances '**Servo**' a échoué en raison d'une **implémentation** défectueuse '**Servo :: Servo (Servo & tocpy)**' copy-constructor.
- 7) Les valeurs hors limites '**Servo.writeMicroseconds()**' ont été correctement détectées comme une erreur, mais les valeurs limites indiquées accompagnant le texte du message d'erreur étaient erronées.
- 8) Un débit en bauds légal de 115200 n'a pas été accepté lorsqu'il est chargé à partir d'un fichier texte '**I / O Dispositifs**'.
- 9) Les conflits de broche électrique provoqués par un dispositif Curseur Analogique connecté n'ont pas toujours été détectés.
- 10) Dans de rares cas, le passage d'un pointeur de chaîne défectueux (avec la chaîne 0-terminator manquante) à une fonction '**String**' pourrait provoquer un crash de UnoArduSim.
- 11) Le **Volet de Code** pourrait mettre en évidence la ligne d'erreur d'Analyse actuelle dans le **mauvais** module de programme (lorsque '**#include**' a été utilisé).

- 12) Le chargement d'un fichier Dispositifs 'I / O' dont un dispositif pilotait (incorrectement) contre 'Uno' pin 13 a provoqué un blocage du programme dans la fenêtre contextuelle du message d'erreur.
- 13) UnoArduSim avait par erreur autorisé l'utilisateur à coller des caractères non hexadécimaux dans les fenêtres de tampon TX expansées pour SPISLV et I2CSLV.
- 14) Les initialisations de ligne de déclaration ont échoué lorsque la valeur de droite était la valeur `'return'` d'une fonction membre d'objet (comme dans `'int angle = myservo1.read();'`).
- 15) Les variables `'static'` membres ayant des préfixes explicites `'ClassName ::'` n'étaient pas reconnues si elles apparaissaient au tout début d'une ligne (par exemple, dans une affectation à une variable base-'class'),
- 16) Appeler `'delete'` sur un pointeur créé par `'new'` n'a été reconnu que si la notation parenthèse de fonction a été utilisée, comme dans `'delete (pptr)'`.
- 17) L'implémentation UnoArduSim de `'noTone()'` insistait à tort sur le fait qu'un argument-pin soit fourni.
- 18) Les modifications ayant augmenté global 'RAM' octets dans un programme utilisant `'String'` variables (via **Éditer / Examiner** ou **Fichier | Charger**), peuvent entraîner une corruption dans cet espace 'Uno' global en raison de la suppression par tas des objets `'String'` appartenant à l'ancien programme en utilisant (incorrectement) le tas appartenant au nouveau programme. Dans certains cas, cela peut entraîner un blocage du programme. Bien qu'un second Charger ou Analyser ait résolu le problème, ce bug a finalement été corrigé.
- 19) Les valeurs renvoyées pour `'Wire.endTransmission()'` et `'Wire.requestFrom()'` étaient toutes les deux bloquées à 0 - elles ont maintenant été corrigées.

### **V1.7.2- fév.2017**

- 1) Les interruptions sur la broche 2 étaient également déclenchées (par inadvertance) par l'activité du signal sur la broche 3 (et vice versa).

### **V1.7.1- Fév.2017**

- 1) Fonction `'delayMicroseconds()'` produisait un retard en *millisecondes* (1000 fois trop grand).
- 2) La conversion explicite d'une variable `'unsigned'` en un type entier plus long a donné un résultat incorrect (`'signed'`).
- 3) Les littéraux hexadécimaux supérieurs à `0 x7FFF` sont maintenant `'long'` par définition, et ainsi, ils généreront maintenant des expressions arithmétiques `'long'` dans lesquelles ils sont impliqués.
- 4) Un bogue introduit par inadvertance par V1.7.0 a empêché la conversion en style C++ des littéraux numériques (par exemple, `'(long) 1000 * 3000'` n'a pas été accepté).
- 5) `'Serial'` ne prend plus ses nombreux octets dans 'Uno' RAM si le programme utilisateur n'en a jamais besoin.
- 6) Les variables globales déclarées par l'utilisateur ne prennent plus d'espace dans 'Uno' RAM si elles ne sont jamais réellement utilisées.
- 7) Les variables membres uniques déclarées comme `'const'`, `'enum'`, et les pointeurs vers les littéraux de chaîne, n'occupent plus d'espace dans 'Uno' RAM (pour être en accord avec la compilation Arduino),
- 8) Les octets de RAM requis pour les bibliothèques `'#include'` intégrées correspondent maintenant étroitement aux résultats de la compilation conditionnelle Arduino.
- 9) L'utilisation de `'new'` sur une ligne de déclaration réelle de pointeur a échoué (seulement l'affecter plus tard avec `'new'` a fonctionné).
- 10) Correction d'un bug où une émission "en attente" d'un répertoire de disque SD pouvait provoquer un blocage de programme.

## **V1.7.0- Déc.2016**

0) Un certain nombre de problèmes avec la gestion des interruptions de l'utilisateur ont maintenant été corrigés:

a) Interrompt les arêtes 0 et 1 qui se sont produites pendant une fonction Arduino **bloquée en attente** (comme '**pulseIn()**' , '**shiftIn()**' , '**spiTransfer()**' , '**flush()**' et '**write()**' ) avait causé une défaillance dans le flux d'exécution au retour d'interruption

b) Des copies multiples des variables locales de toute fonction interrompue étaient apparues dans le **Volet de Variables** (une copie par interruption-retour) et cela a été corrigé dans la version 1.6.3, mais les autres problèmes d'interruption sont restés).

c) Fonction '**delayMicroseconds()**' ne créait aucun délai s'il était appelé depuis l'intérieur d'une routine d'interruption d'utilisateur.

d) Les appels bloquant des fonctions telles que '**pulseIn()**' à l' **intérieur d'** une routine d'interruption ne fonctionnaient pas.

1) Un bogue introduit dans la version 1.6.3 provoquait la perte de la mise à jour des valeurs dans le **Volet de Variables** lors de l'exécution lorsque les valeurs changeaient réellement (cela ne se produisait qu'après deux actions **Arrêt** ou menu **VarRafraichir** utilisateur). En outre, lorsqu'un Exécuter-Vers a été effectué après que **Allow Reduction** ait été déclenché, le **Volet de Variables** n'a pas été redessiné de temps en temps (les anciennes valeurs et les variables locales y sont peut-être apparues jusqu'à l'étape suivante).

2) Le comportement en surbrillance du **Volet de Code** de la commande **Un Pas Franchir** pourrait apparaître trompeuse dans '**if()** - **else**' chaînes - qui a maintenant été corrigé (bien que la fonctionnalité d'**Avancer d'un Pas** réelle était correcte).

3) Fonction '**pulseIn()**' avait incorrectement défini le délai d'attente en millisecondes au lieu de microsecondes - il a également été redémarré de manière incorrecte le délai d'attente lorsque les transitions vers les niveaux inactifs et actifs ont été vus pour la première fois.

4) L'utilisation de littéraux HEX entre 0x8000 et 0xFFFF dans les affectations ou l'arithmétique avec les variables '**long**' integer a donné des résultats incorrects en raison d'une extension de signe non vérifiée.

5) Passer ou renvoyer un '**float**' d'un type '**unsigned**' integer ayant une valeur avec MSB = 1 a donné des résultats incorrects en raison d'une mauvaise interprétation '**signed**' .

6) Toutes les fonctions '**bit \_()**' acceptent maintenant aussi les opérations sur les variables de taille '**long**' , et les tests UnoArduSim pour les positions de bits invalides (qui seraient en dehors de la taille de la variable).

7) Une entrée invalide dans la boîte d'édition '**Pulse**' (width) sur un 'PULSER' dispositif a causé une corruption de la valeur 'Period' (jusqu'à ce qu'elle soit corrigée par l'entrée suivante de l'utilisateur 'Period' edit).

8) La suppression d'un appareil 'PULSER' ou 'FUNCGEN' à l'aide du menu Configurer ne supprimait pas son signal périodique de la broche qu'il pilotait (une réinitialisation n'est plus nécessaire).

9) La possibilité d'initialiser un tableau 1-D '**char**' avec une chaîne entre guillemets manquait (par exemple '**char strg [] = "hello";'** ).

10) L'affichage hexadécimal dans les fenêtres expand 'SERIAL' ou 'SFTSER' Monitor a montré le caractère le plus significatif incorrect pour les valeurs d'octets supérieures à 127.



- 11) Les fenêtres de forme d'onde ne reflétaient pas les modifications programmatiques de l'utilisateur faites par '**analogWrite()**' lorsque la nouvelle valeur était de 0% ou de 100% de cycle de service.
- 12) L'implémentation de '**Serial.end()**' a maintenant été corrigé.
- 13) Un fichier **myUnoPrefs.txt** comportant plus de 3 mots sur une ligne (ou des espaces dans le nom de fichier **Dispositifs 'I / O'**) peut provoquer un blocage dû à un pointeur interne défectueux.
- 14) La dernière ligne d'un fichier Dispositifs 'I / O' n'était pas acceptée si elle ne se terminait pas par un saut de ligne .
- 15) L'ajout de plus de quatre curseurs analogiques a causé un bogue silencieux qui a écrasé les pointeurs LED 'I / O'.
- 16) À partir de la version 1.6.0, les échantillons de forme d'onde analogique pour la ***première moitié*** de chaque forme d'onde ***triangulaire*** étaient tous ***nuls*** (en raison d'un bogue dans le calcul de la table de forme d'onde).
- 17) Effectuer un **Exécuter-Vers** répété sur une ligne de point d'arrêt ne nécessite plus plusieurs clics par avance.
- 18) Le passage d'expressions d'adresse à un paramètre de tableau de fonctions n'a pas été accepté par l'analyseur.
- 19) Les fonctions récursives qui ont renvoyé des expressions contenant des références de pointeur ou de tableau ont donné des résultats incorrects en raison de la désactivation des indicateurs "ready" sur ces expressions de composant.
- 20) L'appel de '**class**' member-fonctions via ***une variable de pointeur d'objet ou une expression de pointeur*** ne fonctionnait pas.
- 21) Les fonctions utilisateur qui renvoyaient des objets par valeur **renvoyaient** uniquement leur valeur lors de leur premier appel de fonction ***s'ils renvoyaient*** un objet construit sans nom (comme '**String** ("**dog**")' - lors d'appels suivants, le retour était ignoré en raison d'un "drapeau).
- 22) Il n'y avait pas de sauvegarde pour empêcher la commande **Fenêtres | 'Serial' Monitor à partir de** l'ajout d'un nouvel appareil '**Serial**' alors qu'il n'y avait pas de place pour cela.
- 23) Si l'ajout d'un dispositif à broche fixe (tel que 'SPISLV') a provoqué un conflit de broche, le **rafraîchissement du Volet du Banc de Laboratoire** peut afficher un dispositif "fantôme" dupliqué superposant le dispositif 'I / O' le plus à droite (jusqu'au prochain rafraîchissement) .
- 24) Correction de certains problèmes avec des sons 'PIEZO' non fiables pour les signaux de broche non périodiques.
- 25) Les variables '**PROGMEM**' doivent maintenant être explicitement déclarées comme '**const**' pour être en accord avec Arduino.
- 26) "Aucun espace de tas" a été incorrectement marqué comme une erreur d'exécution quand un '**SD.open()**' n'a pas pu trouver le fichier nommé, ou un '**openNextFile()**' a atteint le dernier fichier dans le répertoire.
- 27) Un bogue d'analyseur n'acceptait pas correctement une accolade de fermeture hors de place '}' .

28) Un bogue avec **les suppressions du Volet de Variables** lors du retour du constructeur de l'objet-membre a été corrigé (le bogue ne s'appliquait qu'aux objets qui eux-mêmes contiennent d'autres objets en tant que membres).

### **V1.6.3- septembre 2016**

- 1) Les variables locales de toute fonction interrompue ne sont pas supprimées du **Volet de Variables** lors de l'entrée de la fonction d'interruption, ce qui entraîne l'apparition de plusieurs copies lors du retour de la fonction d'interruption (et éventuellement une erreur d'exécution ou un plantage).
- 2) Les fenêtres de forme d'onde ne reflétaient pas les modifications programmatiques dans '`analogWrite()`' à un nouveau cycle d'utilisation de 0% ou 100%.
- 3) L'affichage hexadécimal dans la fenêtre expand 'SERIAL' ou 'SFTSER' Monitor affichait le caractère MSB incorrect pour les valeurs d'octet supérieures à 127.

### **V1.6.2- septembre 2016**

- 1) Les appels de fonction effectués avec le mauvais numéro ou type d'arguments n'ont pas généré un message d'erreur d'Analyse approprié (seul le message générique "pas un identificateur valide" est apparu).
- 2) Le bouton de réinitialisation de la **Barre d'Outils** fonctionne désormais de la même manière que le bouton de réinitialisation de carte 'Uno'.
- 3) Le texte d'erreur d'analyse ne disparaît plus après 16 caractères sans afficher d'ellipse.

### **V1.6.1- août 2016**

- 1) Dans V1.6, une version de carte 'Uno' dans le fichier **myArduPrefs.txt** qui différait de la valeur par défaut de la version 2 provoquait une exception au démarrage (en raison d'un événement PIN 13 non initialisé).
- 2) Si vous modifiez la valeur d'une variable en double-cliquant dans le **Volet de Variables**, vous risquez de générer des fenêtres d'erreur "Pas d'allocation de mémoire" (pour les programmes avec '`class`' ) définis par l'utilisateur .
- 3) '`SoftwareSerial`' n'a pas autorisé l'accès à '`write (char * ptr)`' et '`write (byte * ptr, int size)`' fonctionne à cause d'une détection de surcharge de fonction défectueuse.
- 4) Correction d'un problème avec l'inclusion automatique du fichier ".cpp" correspondant pour un .h "type de bibliothèque" isolé '`#include`' .

### **V1.6- Juin 2016**

- 1) En V1.5 indentation automatique sur la touche "Retour" dans **Éditer / Examiner** (lors de la saisie d'une nouvelle ligne) a été perdu.
- 2) La détection des conflits de broche avec les dispositifs externes 'I / O' à forte conduction a été ajoutée sur '`Serial`' pin 1, sur les broches SPI SS, MOSI et SCK, sur les broches I2C SCL et SDA (toutes les '`begin` correspondantes()' est appelée), et sur toute broche '`SoftwareSerial`' TX déclarée .

### **V1.5.1- Juin 2016**

- 1) Dans la version 1.5, les nouvelles couleurs Syntae-Surligner adaptables aux thèmes n'étaient pas correctement réinitialisées à chaque fois que Éditer / Examiner était ouvert, et donc (avec un thème d'arrière-plan blanc) n'étaient correctes qu'une fois toutes les deux secondes.

2) Les interruptions '**RISING**' et '**FALLING**' sensitivités étaient opposées à la polarité réelle du front de déclenchement.

### **V1.5 - Mai 2016**

1) Un bogue introduit dans la version 1.4.1 empêchait de transmettre des littéraux de chaîne vides aux fonctions membres qui attendaient un objet '**String**', comme dans '**mystring1.startsWith** ("**Hey**")' .

2) Un bug dans le **SD** d'origine l'implémentation de UnoArduSim ne permettait que l'accès **SD** en utilisant les appels à '**read()**' et '**write()**' (l'accès via '**Stream**' fonctions était empêché).

3) '**R = 1K**' Les commutateurs de diapositives n'étaient pas redessinés correctement lorsque le curseur était déplacé.

4) "**Annuler**" dans la boîte de dialogue Confirmer l'enregistrement du fichier aurait dû empêcher l'exécution de l'application.

5) Une citation de fermeture manquante ou la fermeture de '>' -parenthesis sur un fichier utilisateur '**#include**' entraînerait un blocage.

6) Correction d'un bug dans la mise en surbrillance de la syntaxe de '**String**' et de l'utilisateur '**class**' ou '**struct**', et mise en surbrillance étendue pour inclure les appels de fonction constructeur.

7) Correction de problèmes mineurs dans **Éditer / Examiner** avec des changements de texte / mise en surbrillance et le bouton Annuler.

### **V1.4.3 - Avril 2016**

1) Utilisation de **Configurer | I / O** Les **dispositifs** permettant d'ajouter de nouveaux dispositifs, puis de supprimer ultérieurement l'un de ces dispositifs nouvellement ajoutés peuvent provoquer un blocage lors de la réinitialisation ou l'arrêt d'un autre dispositif.

2) La modification d'une variable '**String**' en double-cliquant dans le **Volet de Variables** a échoué (la nouvelle '**String**' n'a pas été lue **correctement** ).

3) Les modifications de broches sur les dispositifs '**FUNCGEN**' et '**PULSER**' n'ont pas été reconnues tant que la réinitialisation n'a pas été effectuée pour la première fois.

### **V1.4.2 - mars 2016**

1) V1.4.1 avait introduit un bug d'**Analyse** malheureux qui empêchait les affectations impliquant des objets '**class**' (y compris '**String**').

2) Une correction de bogue incomplète faite dans V1.4.1 a provoqué '**unsigned**' valeurs de type '**char**' à imprimer en tant que caractères ASCII plutôt qu'en tant que leurs valeurs entières.

3) Les arguments d'appel de fonction d'expression de membre complexe n'étaient pas toujours reconnus en tant que correspondances de paramètre de fonction valides.

4) **Tous** les littéraux et expressions entiers ont été dimensionnés trop généreusement (à '**long**') et donc l'exécution n'a pas reflété les dépassements **réels** (à négatifs) qui peuvent se produire dans Arduino sur les opérations add / multiply impliquant des valeurs '**int**' .

5) Les expressions impliquant une combinaison des types '**signed**' et '**unsigned**' integer n'étaient pas toujours gérées correctement (la valeur '**signed**' ne serait pas considérée comme '**unsigned**').

6) Dans les cas de conflit d'épingles, les messages d'erreur "value =" peuvent afficher des valeurs d'épingle périmées même après une réinitialisation d'un conflit précédent que l'utilisateur avait déjà effacé.

#### **V1.4.1 - Janv. 2016**

- 1) Les appels à '**print (char)**' s'impriment correctement sous la forme de caractères ASCII (plutôt que de valeurs numériques).
- 2) La réponse d'interruption est maintenant activée par défaut lorsque '**attachInterrupt()**' est appelée, donc il n'y a plus besoin de '**setup()**' appeler la fonction d'activation '**interrupts()**' .
- 3) Multiple '**#include**' les instances de fichiers utilisateur à l'intérieur d'un fichier sont maintenant gérées correctement.

#### **V1.4 - Déc. 2015**

- 1) Un bogue de **longue date** indiquait incorrectement une condition de division par **zéro** lors d'une division par une valeur inférieure à l'unité.
- 2) Correction de '**SoftwareSerial**' (qui a été interrompu par inadvertance par une vérification de validation '**class**' - member ajoutée dans les versions V1.3).
- 3) Les appels de fonction de fin de ligne avec un point-virgule manquant n'ont pas été interceptés et l'analyseur a ignoré la ligne suivante.
- 4) Un fichier texte Dispositifs 'I / O' mal formaté a donné un message d'erreur incorrect.
- 5) La mise en évidence de l'erreur d'analyse de la ligne incorrecte (adjacente) dans les expressions et les instructions multi-lignes a été corrigée
- 6) Test logique des pointeurs en utilisant le '**not**' ( **!** ) l'opérateur a été inversé.

#### **V1.3 - Oct. 2015**

- 1) Une mauvaise gestion interne des variables du bloc-notes a occasionnellement occasionné une "**profondeur maximale d'imbrication du bloc-notes dépassée**". Erreurs d'analyse.
- 2) Les parenthèses à l' *intérieur des guillemets simples* , des accolades, des points-virgules, des parenthèses à l' intérieur de chaînes entre guillemets et des caractères échappés ont été mal gérées.
- 3) Un tableau avec une dimension vide et aucune liste d'initialisation a provoqué un blocage de Réinitialiser, et des tableaux avec un seul élément n'ont pas été refusés (et ont provoqué leur interprétation erronée comme un pointeur invalide initialisé).
- 4) Les erreurs d'analyse parfois mettent parfois en évidence la mauvaise ligne (adjacente).
- 5) Passer un pointeur vers un non 'const' à une fonction acceptant un pointeur vers '**const**' avait été refusé (au lieu de l'inverse).
- 6) Les expressions Initializer héritent incorrectement des qualificatifs '**PROGMEM**' de la variable en cours d'initialisation
- 7) Les variables '**PROGMEM**' déclarées avaient leur taille d'octet comptée de manière incorrecte **deux fois par** rapport à leur allocation 'Flash' memory pendant l'analyse.

8) Taper dans 'Send' boîte d'éditer d'un 'I2CSLV' provoquait parfois un crash dû à 'scanf' bug.

9) Le chargement d'un nouveau programme comportant un nouveau fichier Dispositifs 'I / O' dans son répertoire peut entraîner des conflits de broche non pertinents avec les anciennes directions des broches.

10) La gestion des caractères de série échappés a été appliquée de manière incorrecte aux séquences de caractères reçues plutôt que transmises dans la fenêtre (plus grande) 'Serial Monitor' buffers.

11) 'while()' et 'for()' boucles avec des corps complètement vides, tels que 'while (true);' ou 'for (int k = 1; k <= 100; k ++);' passé l'analyse (avec un message d'avertissement) mais a échoué au moment de l'exécution.

## **V1.2 - juin 2015**

1) La très simple des fonctions utilisateur qui a fait des appels à 'digitalRead()' ou 'analogRead()' ou 'bit()' aurait pu corrompre leur (très première) variable locale déclarée (le cas échéant) en raison d'une fonction allouée insuffisante. espace (si seulement deux octets de bloc-notes ont été alloués au tout début de la pile de fonctions). Toute expression numérique à l'intérieur d'une fonction est suffisante pour provoquer une allocation de bloc-notes de 4 octets, évitant ainsi ce problème. Ce bug malheureux a été autour depuis la version originale V1.0.

2) Des fonctions qui sont 'void' avec un explicite 'return', et des fonctions non-'void' avec plus d'un déclaration 'return', verrait l'exécution tomber à travers l'accolade de fermeture (si ca 'return' a été atteint).

3) Toutes les déclarations 'return' dans les contextes 'if()' qui manquaient d'accolades ont conduit à une cible de retour à l'appelant défectueuse.

4) 'PULSER' et 'FUNCGEN' Les largeurs d'impulsion ou les périodes de valeur 0 peuvent provoquer un plantage (0 n'est plus autorisé).

5) Où il n'y avait pas d'accolades, 'else' continuations après un 'if()' ne fonctionnait pas s'ils suivaient un 'break', 'continue', ou 'return'.

6) Lorsque **plusieurs déclarations d'utilisateur 'enum' ont été faites**, les constantes ont été définies dans toutes les **erreurs d'analyse "enum mismatch"** qui ont été générées 'enum' (ce bug a été introduit dans la version 1.1).

7) Un identificateur null pour le tout dernier paramètre d'un prototype de fonction a provoqué une erreur d'Analyse.

8) **Les points d'arrêt Exécuter-Vers** définis sur des lignes complexes n'étaient pas toujours gérés correctement (et pouvaient donc être manqués).

9) 'HardwareSerial' et 'SoftwareSerial' a utilisé un tampon TX-pending d'implémentation privée qui n'a pas été nettoyé lors de la réinitialisation (les caractères restants de la dernière fois pouvaient donc apparaître).

10) L'analyse n'a pas réussi à vérifier la **permutation** des bits illégaux de 'float' et l'arithmétique du pointeur a été tentée avec des opérateurs illégaux.

## **V1.1 - mars 2015**

1) Les index de tableau qui étaient des variables de taille 'byte' ou 'char' ont provoqué des décalages de tableau incorrects (si une variable adjacente contenait un octet haut non-0).

- 2) Le test logique des pointeurs a testé la valeur point-to pour non-zéro plutôt que la valeur du pointeur elle-même.
- 3) Toutes les instructions `'return'` incorporées dans `'for()'` ou `'while()'` boucles ont été mal gérées .
- 4) Les listes d'initialisation d'agrégat pour les tableaux d'objets ou les objets contenant d'autres objets / tableaux ou des listes d'initialisation complètement vides n'étaient pas gérées correctement.
- 5) Accès aux valeurs `'enum'` member à l'aide d'un `" nomenumère ."` Le préfixe n'était pas supporté.
- 6) L'initialisation de la ligne de déclaration d'un `tableau 'char []'` avec un littéral de chaîne entre guillemets ne fonctionnait pas.
- 7) Un tableau transmis à une fonction sans initialisation préalable a été incorrectement signalé avec une erreur "used mais non initialized".
- 8) Les expressions de pointeur impliquant des noms de tableau ont été mal gérées.
- 9) Les paramètres de fonction déclarés comme `'const'` n'ont pas été acceptés.
- 10) La fenêtre Forme d'Onde Analogique n'affiche pas les signaux PWM ( `'servo.write()'` et `'analogWrite()'` ).
- 11) Les fonctions membres accessibles via un pointeur d'objet donnaient des accès membres défectueux.
- 12) Les formes d'onde n'étaient pas mises à jour lorsqu'un point d'arrêt **Exécuter-Vers** était atteint.
- 13) La modélisation d'allocation de registre peut échouer lorsqu'un paramètre de fonction est utilisé directement comme argument d'un autre appel de fonction

#### **V1.0.2 - août 2014**

Ordre fixe des broches A0-A5 sur le périmètre de la carte 'Uno' .

#### **V1.0.1 - Juin 2014**

Correction d'un bug qui tronquait les pentes d'édition qui étaient plus de trois fois le nombre d'octets dans le programme d'origine.

#### **V1.0 - première version mai 2014**

## Changements / Améliorations

### V2.0.1- Janv. 2018

- 1) Les fonctions Arduino non documentées '**exp()**' et '**log()**' ont maintenant été ajoutées.
- 2) Les dispositifs 'SERVO' peuvent maintenant être rendus en rotation continue (la largeur d'impulsion contrôle donc la vitesse au lieu de l'angle).
- 3) Dans **Éditer / Examiner**, une accolade fermante '**}**' est maintenant ajoutée automatiquement lorsque vous tapez une accolade ouvrante '**{**'.
- 4) Si vous cliquez sur '**X**' dans la barre de titre de la fenêtre **Éditer / Examiner** pour quitter, vous avez maintenant la possibilité d'annuler si vous avez modifié, mais pas enregistré, le fichier programme affiché.

### V2.0 Déc. 2017

- 0) L'implémentation a été portée sur QtCreator de sorte que l'interface graphique présente des différences visuelles mineures, mais aucune différence fonctionnelle autre que certaines améliorations:
  - a) La messagerie de la ligne d'état au bas de la fenêtre principale et à l'intérieur de la boîte de dialogue **Éditer / Examiner** a été améliorée et un code de couleurs en surbrillance a été ajouté.
  - b) L'espace vertical alloué entre le **Volet de Code** et le **Volet de Variables** est maintenant ajustable grâce à une barre de division déplaçable (mais non visible) à leur frontière commune.
  - c) Les valeurs dans les boîtes d'édition d'un dispositif '**I / O**' ne sont désormais validées que lorsque l'utilisateur a déplacé le pointeur de la souris hors de l'appareil - ceci évite les changements automatiques gênants pour appliquer les valeurs légales pendant que l'utilisateur tape.
- 1) UnoArduSim prend désormais en charge plusieurs langues via **Configurer | Préférences. L'anglais** peut toujours être sélectionné, en plus de la langue des paramètres régionaux de l'utilisateur (à condition qu'un fichier de traduction \*.qm personnalisé pour cette langue soit présent dans le dossier UnoArduSim '**translations**').
- 2) Le son a maintenant été modifié pour utiliser l'API audio Qt - ceci a nécessité une mise en sourdine dans certaines circonstances afin d'éviter des bruits de claquement et des clics gênants pendant les longs délais opérationnels de fenêtrage du système d'exploitation provoqués par des clics de souris normales - voir la section Modélisation détail à ce sujet.
- 3) Pour plus de commodité, les blancs sont maintenant utilisés pour représenter une valeur 0 dans les zones d'édition de nombre de dispositifs dans **Configurer | Dispositifs 'I / O'** (vous pouvez maintenant utiliser la barre d'espace pour supprimer des dispositifs).
- 4) Le qualificatif non mis à l'échelle (U) est désormais facultatif sur 'PULSER', 'FUNCGEN' et '1SHOT' dispositifs (il s'agit de la valeur par défaut).
- 5) UnoArduSim permet maintenant (en plus des valeurs numériques littérales) '**const**' valeurs variables entières, et '**enum**' members, d'être utilisées comme dimensions dans les déclarations de tableau, tant que ces sources sont initialisées explicitement en utilisant un littéral numérique entier constantes.
- 6) Après l'entrée de la souris, un dispositif PUSH produira un rebond de contact (pendant 1 milliseconde) à chaque pression sur la touche de barre d'espace (mais pas pour les clics de souris, ni pour aucune autre touche).
- 7) Clics supplémentaires de (Trouver | Définir le Texte de la Recherche) sélectionne maintenant le **suivant** mot à partir du texte de la ligne en surbrillance dans le volet actif (le Volet de Code ou le Volet de Variables).
- 8) Fonctions '**max()**' et '**pow()**' **ont maintenant été inclus dans la** commodité liste des Built-ins dans la boîte de dialogue **Éditer / Examiner**.



9) La commande 'goto' Arduino est maintenant signalée comme "non supportée" dans UnoArduSim.

### **V1.7.2- février 2017**

1) Le choix de couleur bleu (B) a été ajouté pour les appareils à LED.

### **V1.7.1- février 2017**

1) Les suffixes 'L' et / ou 'U' sont maintenant acceptés à la fin des constantes littérales numériques (pour les définir comme 'long' et / ou 'unsigned' ), et les constantes binaires ( 0b ou 0B ) sont maintenant acceptées. Toute constante numérique tout décimal **commençant par '0'** est maintenant considérée comme une valeur **octale** . (d'accord avec Arduino).

2) Lors de l'exécution dans une boucle serrée à partir de laquelle il n'y a pas d'échappement (par exemple 'while(x) ; x++;' où x est toujours vrai), cliquez sur **Arrêter** une seconde fois pour **arrêter l'** exécution du programme (et sur cette ligne de programme défectueuse) .

### **V1.7.0- Déc. 2016**

1) Une nouvelle fonctionnalité de la **Barre d'Outils** a été ajoutée qui affiche des octets de RAM gratuits pendant l'exécution du programme (en tenant compte du nombre total d'octets utilisés par les variables globales, les allocations de tas et les variables de pile locales).

2) Les fonctions d'interruption de l'utilisateur peuvent maintenant elles-mêmes appeler des fonctions Arduino bloquantes comme 'pulseIn()' (mais ceci ne doit être utilisé qu'avec prudence, car la fonction d'interruption ne retournera pas tant que la fonction de blocage n'est pas terminée).

3) Les interruptions de l'utilisateur ne sont plus désactivées lors des opérations de lecture de flux bloquées. Par conséquent, le comportement correspond à l'opération de lecture de flux Arduino.

4) Vous pouvez maintenant entrer et sortir du blocage des fonctions Arduino qui peuvent être interrompues (comme 'delay()' et 'pulseIn()' ), et les messages de la barre d'état ont été augmentés pour montrer quand vous avez frappé un point d'arrêt d'interruption à l'intérieur d'une telle fonction (ou lorsque vous cliquez sur-Arrêter lorsque l'exécution est actuellement dans une telle fonction).

5) Une nouvelle commande **Exécuter-Jusqu'à** (et une **Barre d'Outils** ) a été ajoutée - un simple clic sur n'importe quelle **variable de** Variables (simple, agrégat ou objet, ou élément de tableau ou membre d'objet) pour le mettre en évidence, alors **Exécuter-Jusqu'à** - l'exécution se bloquera au prochain **accès en écriture à l'** intérieur de cette variable agrégée, ou à cet emplacement unique.

6) Lorsque l'exécution se fige après une action **Avancer d'un Pas** , **Exécuter-Vers** , **Exécuter-Jusqu'à** ou **Exécuter-puis-Arrêter** , le **Volet de Variables** met en surbrillance la variable correspondant aux **emplacements d'adresse qui ont été modifiés** (le cas échéant) par le **dernier instruction Au cours de cette exécution** - si cet emplacement est actuellement caché à l'intérieur d'un tableau ou d'un objet non-développé, cliquer pour le développer entraînera alors la mise en surbrillance de cet élément ou membre modifié en dernier.

7) L'utilisateur peut maintenant garder une montre spéciale sur la valeur d'une variable de Volet de Variables spécifique / membre / élément lors de l' exécution - double-cliquez sur cette ligne dans le **Volet de Variables** pour ouvrir le **menu Modifier / Surveiller** fenêtre de **valeur variable**, puis effectuez l' une des **Exécuter** ou Commandes d' **étape** - la valeur montrée sera mise à jour pendant l'exécution selon les mêmes règles qui régissent les mises à jour dans le **Volet de Variables** . Après l'arrêt de l'exécution, vous êtes autorisé à entrer une nouvelle valeur et à l' **accepter** avant de reprendre l'exécution (et vous pouvez **revenir** à la valeur pré- **Acceptée** si vous changez d'avis avant cette date).

- 8) Les touches d'accélérateur F4-F10 ont été réglées pour correspondre aux commandes de la **barre d'outils** du menu Exécuter (de gauche à droite).
- 9) En double-cliquant dessus, cliquer avec le bouton droit sur 'SERIAL', 'SFTSER', 'SPISLV', 'I2CSLV' les dispositifs vont maintenant afficher une fenêtre TX / RX octets / chars de plus grande taille (et sur 'SD\_DRV', une fenêtre de surveillance de fichiers).
- 10) La zone d'édition TX dans 'SERIAL' ou 'SFTSER' n'est plus désactivée lors d'une transmission de caractères active (vous pouvez maintenant ajouter ou remplacer ce qui existe), mais un bouton de retour chariot (ou 'Send' dans la fenêtre associée 'SerialMonitor' child) sera ignorée jusqu'à ce que la transmission revienne à l'état inactif (les caractères sont maintenant affichés en *italique* lorsque la transmission est prête à commencer, et est actif). En outre, l'utilisateur est maintenant averti à un flux en série '**begin()**' s'ils avaient déjà démarré plus tôt les transmissions de dispositifs connectés (en cours), car il n'y aurait alors pas de synchronisation de trame, ce qui conduirait à des erreurs de réception.
- 11) Le **délai** par défaut ajouté '**loop()**' a été augmenté de 250 microsecondes à une milliseconde afin de ne pas tomber aussi loin derrière le temps réel lorsque l'utilisateur néglige d'inclure '**delay()**' (explicite ou naturel) quelque part à l'intérieur '**loop()**' ou à l'intérieur d'une fonction qu'il appelle.
- 12) Les tableaux et les types simples ont maintenant été ajoutés au support de l'instruction heap-allocation '**new**'.
- 13) Des vérifications plus approfondies (et des messages d'erreur associés) ont été ajoutées pour les accès aux adresses hors limites du programme utilisateur (c'est-à-dire en dehors de 'Uno' RAM, ou en dehors de 'Flash' pour '**PROGMEM**' **accesses**).
- 14) Les valeurs du pointeur dans le **Volet de Variables** ressemblent maintenant davantage aux valeurs réelles du pointeur Arduino.
- 15) Le fichier **myArduPrefs.txt** utilisateur est maintenant chargé à chaque **Fichier | Chargez**, pas seulement au lancement d'UnoArduSim.
- 16) Une erreur d' **analyse** est maintenant marquée en essayant de '**attachInterrupt()**' à une fonction utilisateur qui n'est pas '**void**' retourner, ou qui a des paramètres de fonction, ou qui n'a pas été déclaré quelque part avant '**attachInterrupt()**'.
- 17) '**static**' variables-membres sont maintenant affichées en haut du **Volet de Variables** sous forme de globales, plutôt que d'apparaître dans chaque instance d'un objet (développé).
- 18) Fonction '**availableForWrite()**' a été ajouté à l'implémentation de `Serial`.
- 19) Tous les spéciaux '**PROGMEM**', '**typedef**' comme '**prog\_char**' et '**prog\_int16**' ont maintenant été supprimés (ils ont été abandonnés dans Arduino).
- 20) Amélioration des messages d'erreur pour les erreurs d'analyse provoquées par des types de déclaration mal orthographiés ou invalides.
- 21) La taille maximale autorisée du programme a été augmentée.

### **V1.6.3- septembre 2016**

- 1) Ajout d'un message d'erreur d'analyse améliorée lorsque '**attachInterrupt()**' fait référence à une fonction d'interruption qui n'a pas été **prototypée auparavant**.

2) Ajout d'un message d'erreur d'Analyse amélioré pour les listes d'initialisation de tableau multidimensionnel.

### **V1.6.2- septembre 2016**

1) Ajout d'un contrôle d'édition Trouver-Texte à la barre d' outils pour rationaliser la recherche de texte (dans le **Volet de Code** et dans le **Volet de Variables** ).

2) Le bouton de réinitialisation de la **Barre d'Outils** fonctionne désormais de la même manière que le bouton-poussoir de réinitialisation 'Uno' carte.

### **V1.6.1- août 2016**

Ajout d'une vérification pour éviter le chargement et l'analyse en double des fichiers déjà `'#include'` .

### **V1.6 - Juin 2016**

1) Ajout d'un nouveau '1SHOT' (Un-Tir) dispositif 'I / O' qui génère une impulsion après un délai choisi à partir d'un front de signal de déclenchement de polarité sélectionnée.

2) Ajout d'une nouvelle fonctionnalité pour rendre les valeurs de boîte d'édition 'I / O' facilement **évolutives** pendant l'exécution en faisant glisser un curseur global 'I / O \_\_\_\_\_ S' Scaler sur la **Barre d'Outils** principale (il suffit de taper 's' ou 'S' après une valeur pour indiquer la mise à l'échelle).

### **V1.5.1 - Juin 2016**

1) Le support a été ajouté pour les fonctions de la librairie EEPROM `'update()'` , `'put()'` et `'get()'` , et pour l'accès byte via la notation de tableau, par ex. `'EEPROM [k]'` .

2) **Allow Auto (-) Collapse** a été ajouté au menu **VarRafraichir** pour permettre un contrôle explicite sur la possibilité d'effondrement automatique des tableaux / objets développés lorsque l'exécution est en retard sur le temps réel.

3) Les personnages d'un `'String'` variable peut maintenant être également consulté via la notation de tableau, par exemple `'mystring [k]'` .

### **V1.5 - Mai 2016**

1) **Éditer / Examiner a** maintenant raccourci ctrl-E, et a un nouveau bouton pour **Compiler** (ctrl-R), plus une case intégrée d'erreur d'analyse, pour permettre le test des modifications sans avoir besoin de fermer la fenêtre.

2) **Éditer / Examiner** maintenant désormais en charge **Refaire** , et dispose d'un nouveau bouton **Enregistrer** (ctrl-S) (équivalent à **Accept** plus une **sauvegarde de la** fenêtre principale ), et donne maintenant un choix de taille TAB (une nouvelle préférence qui peut être sauvegardée en utilisant **Configurer | Préférences** ). {/0

3) Toutes les zones d'édition modifiables suivent désormais les couleurs de thème Windows OS choisies et, pour le contraste, toutes les zones d'édition en lecture seule 'RECV' edit utilisent du texte blanc sur fond noir. Les **couleurs d'arrière-plan Éditer / Examiner et Syntaxe-highlight s'adaptent désormais également au thème choisi**.

4) UnoArduSim permet maintenant un choix de police - ce choix, et sa taille, ont été déplacés dans le **Configurer | Préférences** (peuvent donc être sauvegardées dans le fichier **myArduPrefs.txt** ).

5) Les valeurs littérales binaires prédéfinies Arduino (comme `'B01011011'` ) sont désormais autorisées.

- 6) Les séquences hexadécimales hexadécimales, octales et à 4 chiffres Unicode peuvent désormais être utilisées comme littéraux numériques.
- 7) Après un premier clic de souris sur un pavé numérique 'PUSH', l'utilisateur peut alors utiliser une touche (n'importe quelle touche) pour enfoncer les contacts du bouton-poussoir.
- 8) **Éditer / Examiner** libère maintenant son état initial en lecture seule temporaire (et supprime la mise en surbrillance de la ligne sélectionnée initiale) après un bref signal visuel instantané.
- 9) UnoArduSim vérifie maintenant les multiples conflits 'Stepper' et 'Servo' pin, c'est-à-dire que le programme utilisateur défectueux tente de se connecter aux broches déjà attachés aux **variables** 'Stepper' ou 'Servo' antérieures.
- 10) Une erreur d'Analyse provoquée par un côté gauche ou droit manquant à un opérateur (manquant une expression ou une variable LHS ou RHS) génère maintenant un message d'erreur clair.
- 11) La variable 'String' class 'flags' membre non utilisée a été supprimée pour être en accord avec Arduino V1.6.6. Un objet 'String' occupe maintenant 6 octets (plus son allocation de tas de caractères).

### **V1.4.2 - mars 2016**

- 1) Les fonctions définies avant (c. -à-d. Celles sans déclaration de prototype avant leur premier appel) génèrent maintenant des avertissements (pas des erreurs d'analyse) lorsque la définition de fonction ultérieure return-type ne correspond pas au type déduit de leur première utilisation.
- 2) Les tableaux ayant une dimension égale à 1 ne sont plus rejetés (pour être en accord avec les règles C++ standard).
- 3) Les boîtes d'édition ne sont plus définies en noir sur fond blanc - elles adoptent maintenant la palette définie par le thème Windows OS utilisé.
- 4) 'SERIAL', 'SFTSER', 'SPISLV', et 'I2CSLV' dispositif élargi Les fenêtres du moniteur (ouvertes en double-clic) adoptent maintenant la couleur d'arrière-plan de leur parent dispositif 'I / O'.

### **V1.4 - décembre 2015**

- 1) La fonctionnalité 'Stepper.h' **bibliothèque et les dispositifs 'I / O' associés ont été ajoutés.**
- 2) Tous les paramètres et valeurs d'un dispositif 'I / O' (en plus des broches sélectionnées) sont maintenant enregistrés dans le fichier texte Dispositifs 'I / O' choisi pour un rechargement ultérieur.
- 3) La couleur de la **LED** 'I / O' peut maintenant être définie en rouge, jaune ou vert à l'aide d'une boîte d'édition sur l'appareil.
- 4) Les initialiseurs de déclaration de variable sont désormais autorisés à s'étendre sur plusieurs lignes.
- 5) Les index de tableau sont maintenant autorisés à être eux-mêmes des éléments de tableau.
- 6) **Configurer | Les préférences** incluent désormais une case à cocher permettant d'utiliser 'and', 'or', 'not' keywords à la place de la norme C &&, || et ! Opérateurs logiques.
- 7) **"Montrez le Téléchargement du Programme" a été déplacé vers Configurer | Préférences**

### **V1.3 - oct 2015**

- 1) Le dispositif 'PUSH de' a maintenant une case à cocher bouton-similaire marqué 'latch' pour les faire "verrouillage" ( au lieu de "momentanée"), qui est, ils se verrouillent en position fermée (et changer de couleur) lorsque appuyez sur, jusqu'à ce qu'ils soient pressés à nouveau pour libérer les contacts.
- 2) Une fonctionnalité complète 'SPISLV' a été ajoutée avec la sélection de noeuds ( 'MODE0' , 'MODE1' , 'MODE2' ou 'MODE3' ). Un double-clic ouvre une fenêtre de tampons TX / RX dans laquelle les octets REPLY (TX) à venir peuvent être définis, et pour visualiser les octets reçus (RX). Le simple dispositif esclave de registre de décalage de la version précédente a été renommé pour devenir un 'SRSLV' dispositif.
- 3) La police de caractère **gras** peut maintenant être choisie pour le **Volet de Code** et le **Volet de Variables** (du menu **Options** ), et la ***mise en évidence audacieuse des mots-clés et des opérateurs*** peut maintenant être activé / désactivé dans **Éditer / Examiner** .
- 4) UnoArduSim autorise maintenant 'bool' comme synonyme de 'boolean' .
- 5) Pour plus de clarté dans les rapports d'erreurs, les déclarations de variables ne sont plus autorisées à s'étendre sur plusieurs lignes (à l'exception des tableaux ayant des listes d'initialisation).
- 6) La vitesse de coloration de la syntaxe dans **Éditer / Examiner** a été améliorée (ceci sera visible avec les programmes plus grands).
- 7) Un surcoût optionnel de 200 microsecondes (dans le menu **Options** ) a été ajouté à chaque appel de 'loop()' - pour éviter de tomber trop loin derrière le temps réel dans le cas où le programme utilisateur n'a pas ajouté 'delay ( )' n'importe où (voir Discussion sur la synchronisation sous **Modélisation** ).

## **Version 1.2 Juin 2015**

- 1) La bibliothèque SD est maintenant entièrement implémentée et un (petit) disque SD 'I / O' de 8 Mo ('SD\_DRV') a été ajouté (et la fonctionnalité a été testée sur tous les exemples de programmes SD d'Arduino).
- 2) Comme Arduino, UnoArduSim va maintenant convertir automatiquement un argument de fonction à son adresse lors de l'appel d'une fonction qui attend qu'un pointeur soit passé.
- 3) Les messages d'erreur d'analyse sont désormais plus appropriés lorsque des points-virgules sont manquants et après des déclarations non reconnues.
- 4) Les surlignés périmées de la ligne du **Volet de Variables** sont désormais supprimés lors de l'appel / ou retour d'une fonction.

## **V1.1 - mars 2015**

- 1) La fenêtre principale peut maintenant être agrandie ou **redimensionnée** pour agrandir le **Volet de Code** et le **Volet de Variables** (pour les écrans plus grands).
- 2) Un nouveau menu Rechercher (avec les **boutons de la barre d' outils** ) a été ajouté pour permettre une navigation plus rapide dans le **Volet de Code** et les **Volet de Variables** (PgUp et PgDown, ou recherche de texte avec flèche vers le haut, flèche vers le bas).
- 3) La fenêtre **Éditer / Examiner** permet maintenant les sauts de navigation ctrl-PgUp et ctrl-PgDn (jusqu'à la prochaine ligne vide), et a augmenté la fonctionnalité **Trouver / Remplacer** .

- 4) Un nouvel élément a été ajouté au menu **VarRafraichir** pour permettre à l'utilisateur de sélectionner une approche d'économie de calcul sous les lourdes charges de mise à jour du **Volet de Variables** .
- 5) Les broches 'Uno' et la LED attachée reflètent maintenant les changements apportés aux dispositifs 'I / O' même lorsque le temps est gelé (c'est-à-dire même lorsque l'exécution est arrêtée).
- 6) D'autres fonctions utilisateur peuvent désormais être appelées à partir d'une fonction d'interruption utilisateur (conformément à la mise à jour vers Arduino 1.06).
- 7) Une **police plus grande** peut maintenant être choisie dans le menu **Options** .

#### **V1.0.1 - Juin 2014**

Les fenêtres de **forme d'onde** indiquent maintenant les broches analogiques comme A0-A5 au lieu de 14-19.

#### **V1.0 - première version mai 2014**