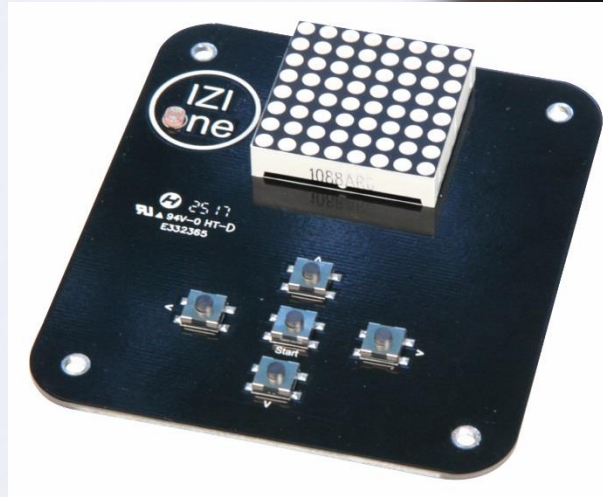


IZIone

Matrice à LED programmable



Manuel utilisateur & Fiches d'activité



```
Arduino - générer le code  
répéter indéfiniment  
mettre Compteur à 10  
répéter 11 fois  
afficher le nombre Compteur  
attendre 1 secondes  
ajouter à Compteur -1
```



Ressources disponibles pour le projet IZlone

Autour du projet IZlone, nous vous proposons un ensemble de **ressources téléchargeables gratuitement sur le wiki** : <http://a4.fr/wiki/index.php/IZlone>

- Un **manuel utilisateur et des fiches d'activités** ;
- Les programmes et extensions correspondants réalisés sous **Scratch mBlock** ;
- Le logiciel de programmation **Scratch mBlock** ;
- Les **fichiers 3D** et les **fichiers STL** du boîtier pour l'impression 3D.



Les documents techniques et pédagogiques signés A4 Technologie sont diffusés librement sous licence Creative Commons BY-NC-SA :

- **BY** : Toujours citer A4 Technologie comme source (paternité).
- **NC** : Aucune utilisation commerciale ne peut être autorisée sans l'accord préalable de la société A4.
- **SA** : La diffusion des documents éventuellement modifiés ou adaptés doit se faire sous le même régime.

Consulter le site <http://creativecommons.fr/>

Nota : la duplication de ce dossier est donc autorisée sans limite de quantité au sein des établissements scolaires, aux seules fins pédagogiques, à condition que soit cité le nom de l'éditeur A4 Technologie.

Produits associés à IZlone

Carte Arduino Uno
(Réf. ARD-A000066)



Câble de programmation USB
(Réf. CABL-IMPUSB-1M)



Bloc alimentation 12V
(Réf. BLOC-ALIM12VDC1A8)



Boîtier I3D
(Réf. IZI-ONE-BT-I3DR)



Table des Matières

Ressources disponibles pour le projet IZlone	2
Introduction.....	2
Prérequis	2
Caractéristiques techniques	3
Mise en œuvre de IZlone	4
Lancer le logiciel mBlock	4
Brancher et connecter votre IZlone	4
Importer les extensions.....	5
Cacher / Développer les extensions.....	7
Supprimer une extension.....	8
Téléverser un programme sur IZlone	8
Fiches d'activité	10
ACTIVITÉ FLASH : Programmer les capteurs	11
ACTIVITÉ FLASH : Programmer une Lettre.....	11
ACTIVITÉ FLASH : Programmer un chiffre	11
ACTIVITÉ FLASH : Programmer un Smiley	11
ACTIVITÉ : Apprendre à programmer un ascenseur - Séance 1	11
ACTIVITÉ : Apprendre à programmer un ascenseur - Séance 1 - Corrigé.....	11
ACTIVITÉ : Apprendre à programmer un ascenseur - Séance 2.....	11
ACTIVITÉ : Apprendre à programmer un ascenseur - Séance 2 - Corrigé.....	11
ACTIVITÉ : Apprendre à programmer un ascenseur - Séance 3.....	11
ACTIVITÉ : Apprendre à programmer un ascenseur - Séance 3 - Corrigé.....	11
ACTIVITÉ : Apprendre à programmer un compte à rebours.....	11
ACTIVITÉ : Apprendre à programmer un compte à rebours - Corrigé.....	11
ACTIVITÉ : Apprendre à programmer un jeu SNAKE - Séance 1	11
ACTIVITÉ : Apprendre à programmer un jeu SNAKE - Séance 1 - Corrigé	11
ACTIVITÉ : Apprendre à programmer un jeu SNAKE - Séance 2	11
ACTIVITÉ : Apprendre à programmer un jeu SNAKE - Séance 2 - Corrigé	11
ACTIVITÉ : Apprendre à programmer un dessin - Séance 1	11
ACTIVITÉ : Apprendre à programmer un dessin - Séance 2	11
ACTIVITÉ : Apprendre à programmer un dessin - Séance 3	11
ACTIVITÉ : Apprendre à programmer un radar de recul - Séance 1	11
ACTIVITÉ : Apprendre à programmer un radar de recul - Séance 1 - Corrigé	11
ACTIVITÉ : Apprendre à programmer un radar de recul - Séance 2	11
ACTIVITÉ : Apprendre à programmer un radar de recul - Séance 2 - Corrigé	11
ACTIVITÉ : Apprendre à programmer un radar de recul - Séance 3	11
ACTIVITÉ : Apprendre à programmer un radar de recul - Séance 3 - Corrigé	11

Introduction

Ce document se décompose en 2 parties :






- un manuel utilisateur pour vous guider dans la prise en main et la programmation de la matrice IZlone avec Scratch mBlock ;
- des fiches d'activité et leurs corrigés avec des niveaux de difficultés progressifs.

Prérequis

Télécharger le logiciel mBlock mBlock

Rendez-vous à l'adresse suivante : <http://www.mblock.cc/download>

IMPORTANT : Télécharger la version correspondant à votre système d'exploitation, Windows ou Mac.

 Windows 7 and above	V3.4.11	 Mac OS	V3.4.11 Latest OSX recommended
 Windows XP	V3.4.2		
 Chrome OS	Use with Chromebooks	 Linux	Supports Debian and Ubuntu Linux

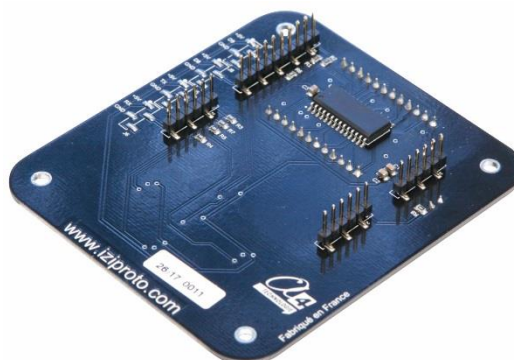
Télécharger les extensions et les programmes

Rendez-vous sur <http://a4.fr/wiki/index.php/lzione>

Par fichier, vous aurez les extensions nécessaires, les fiches activités, les corrigés et des fichiers de départs mBlock. Enregistrez-les sur votre ordinateur.

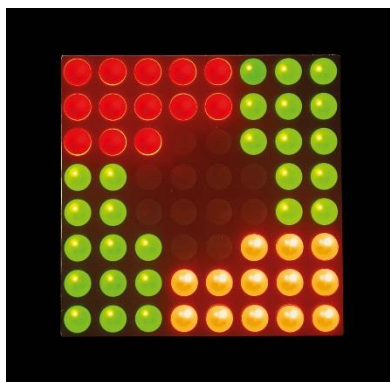
Caractéristiques techniques

IZlone est un shield compatible Arduino, programmable avec Scratch / mBlock et un jeu d'instructions dédiées. C'est un outil facile et ludique pour initier au codage et à la programmation en réalisant toutes sortes d'animations ou de jeux.



Dimensions 80 x 90 mm.

Matrice à LED 32 x 32 mm, 64 points lumineux, 3 couleurs sélectionnables pour chaque LED (rouge/orange/vert).



5 boutons-poussoirs permettent de programmer IZlone en console de jeu.
1 capteur LDR réagit à la lumière ambiante et élargit le champ des activités possibles.

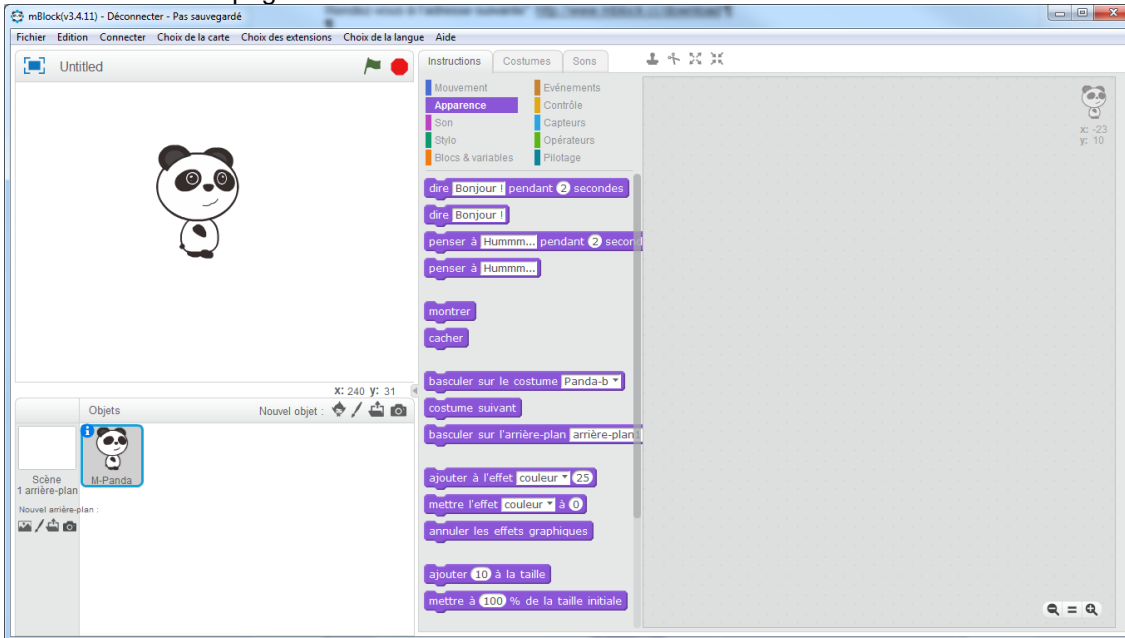


Fonctionne avec une carte Arduino Uno (non fournie).

Mise en œuvre de IZlone

Lancer le logiciel mBlock

Vous arrivez sur la page d'accueil.

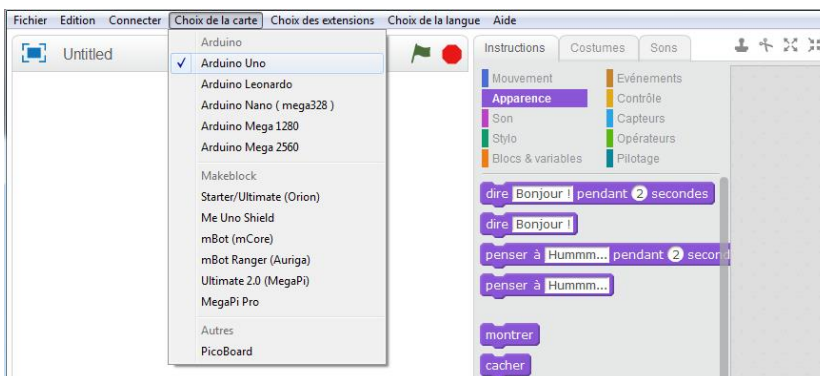


Note : pour modifier la langue d'utilisation, allez dans le menu **Language / Choix de la langue** puis cliquez sur la langue de votre choix.

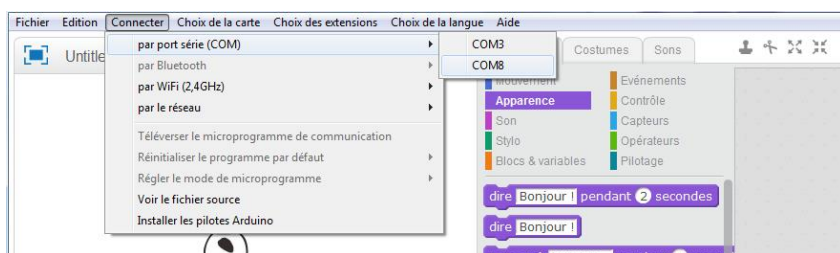
Brancher et connecter votre IZlone

Prenez soin de bien enficher la carte Arduino Uno sur votre IZlone. Les broches de l'IZlone doivent parfaitement s'emboîter dans votre Arduino Uno. Brancher votre IZlone à votre ordinateur via les ports USB. **IMPORTANT** : vérifiez que la LED de l'Arduino Uno s'allume.

Tout d'abord, assurez-vous du bon choix de la carte dans Scratch mBlock. A partir du menu **Choix de la carte**, sélectionnez **Arduino Uno**.

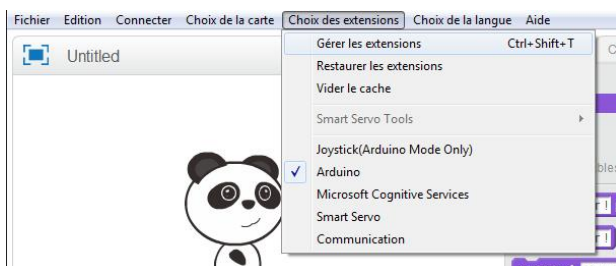


Connectez-vous via le port COM. A partir du menu **Connecter**, sélectionnez **par port série (COM)** puis choisissez le port COM utilisé.

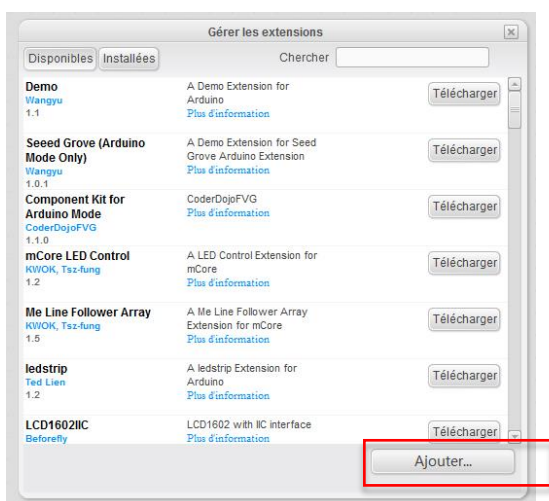


Importer les extensions

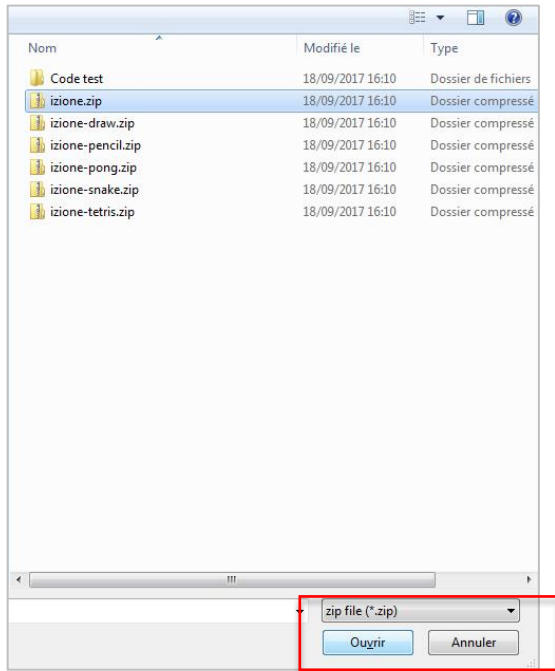
A partir du menu **Choix des extensions**, cliquez sur **Gérer les extensions**.



Une boîte de dialogue s'affiche, cliquez sur **Ajouter...** en bas de la fenêtre.



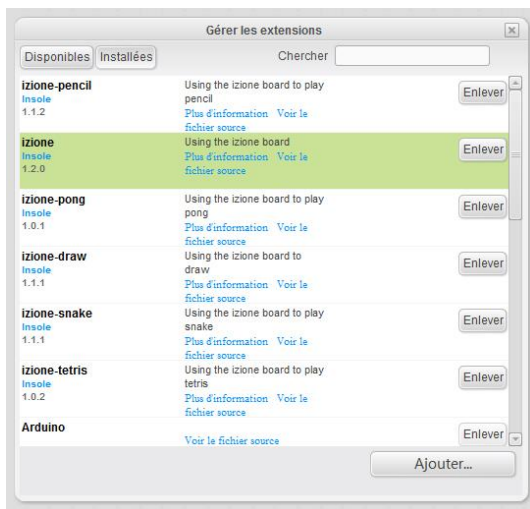
Importez les extensions IZlone depuis l'endroit où vous les avez sauvegardées : cliquez sur **Ouvrir**.



- izione.zip
Il s'agit de l'extension de base, indispensable pour toutes les activités.
- izione-draw.zip
- izione-pencil.zip
permet de réaliser les activités de dessin.
- izione-pong.zip
- izione-snake.zip
- izione-tetris.zip

IMPORTANT : Pour les utilisateurs **Windows**, il se peut que vous ne voyiez pas les extensions à l'endroit précis où les avez sauvegardées. Les extensions sont au format **.Zip file**.

Vous devez répéter l'opération pour chaque extension.

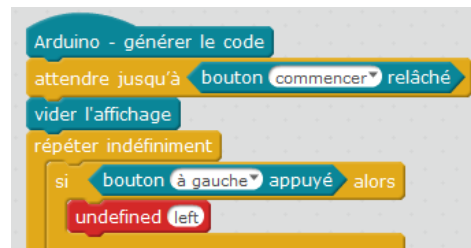
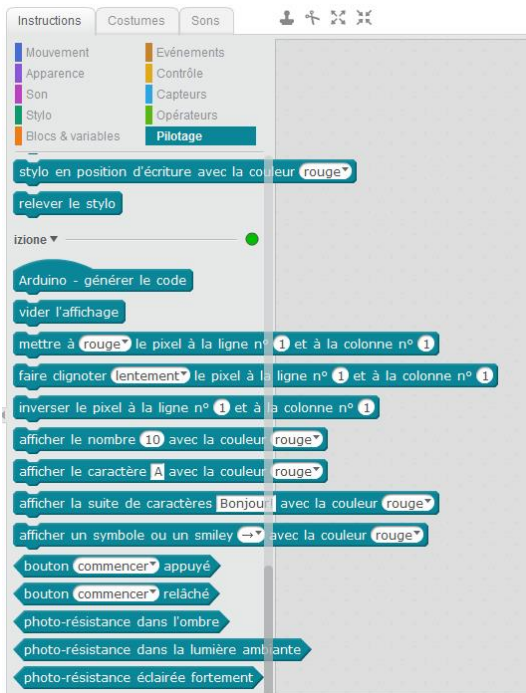


Une fois toutes les extensions importées, fermez la fenêtre de dialogue.

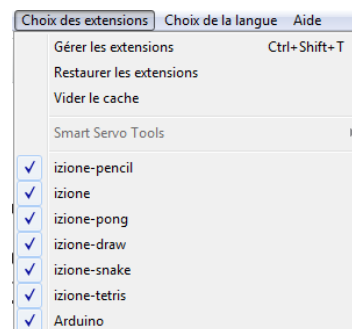
Lorsque vous téléchargez une extension afin de réaliser le programme correspondant, vous devez importer l'extension **izione.zip** avec.

C'est l'extension de base qui vous permet d'interagir avec les « entrées » de l'IZlone, c'est-à-dire avec les boutons.

Vérifiez ensuite que vous avez correctement importé vos extensions. A partir de l'onglet **Instructions**, cliquez sur le menu **Pilotage**.



Il arrive que le bloc rouge « undefined » apparaisse. Vous devez vérifier que l'extension est bien activée à partir du menu **Choix des extensions**.



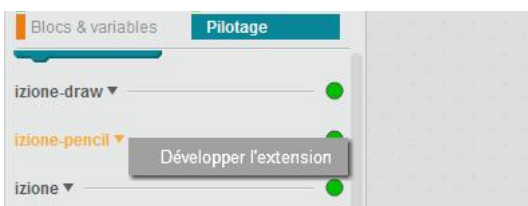
Toutes vos extensions IZlone seront stockées à cet endroit. Cette opération est à faire une seule fois. Si vous fermez et relancez mBlock, vos extensions seront toujours installées.

Cacher / Développer les extensions

Dans un souci de clarté, vous pouvez cacher les extensions si vous en utilisez plusieurs. Cliquez sur une extension. Son nom se met alors en surbrillance orange. Cliquez ensuite sur **Cacher l'extension**.

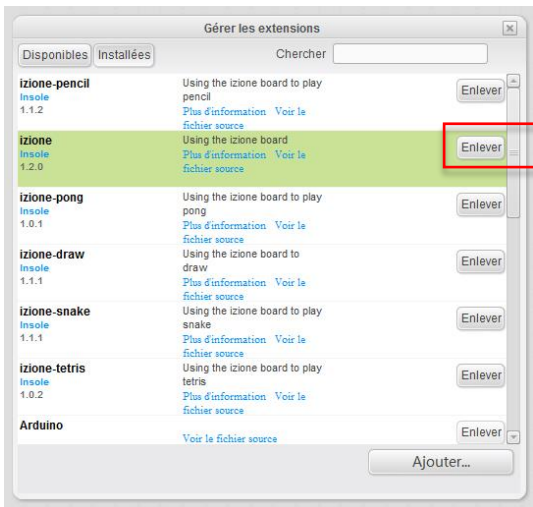


Procédez de la même manière pour faire réapparaître vos extensions. Cliquez sur **Développer l'extension**.



Supprimer une extension

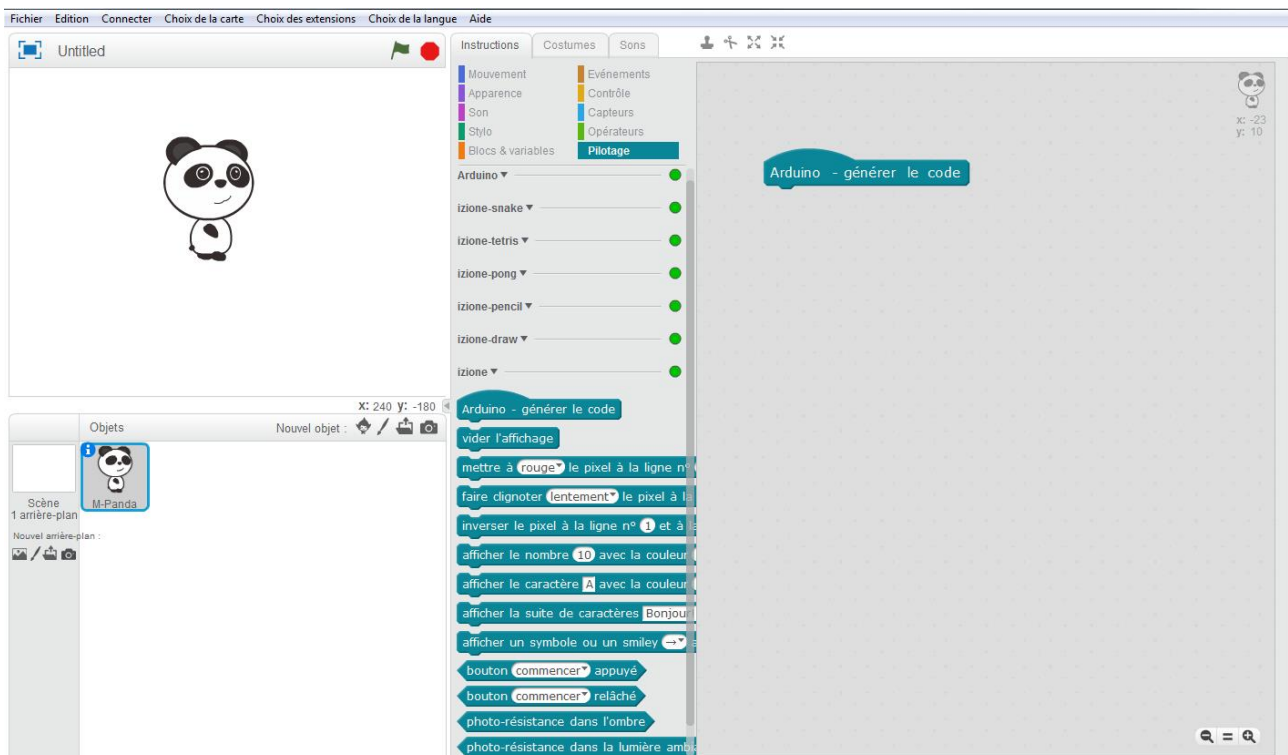
A partir du menu **Choix des extensions**, cliquez sur **Gérer les extensions**.



A partir de l'onglet **Installées**, cliquez que l'extension à supprimer puis sur le bouton **Enlever**.

Téléverser un programme sur IZlone

Réalisez un programme en prenant bien soin de le commencer par le bloc **Arduino - générer le code**. Vous trouverez ce bloc dans l'extension Arduino ou IZlone. Faites un glisser/déposer.



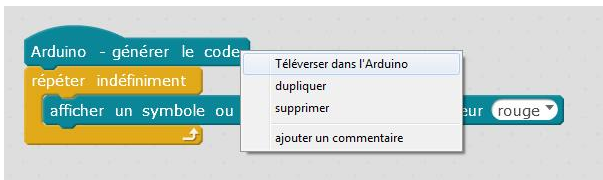
Note : l'extension Arduino se retrouve automatiquement dans vos extensions, car vous avez sélectionné dans : choix de la carte, la carte Arduino Uno.

Développez le programme de votre choix, ici faire apparaître un smiley sur l'écran de votre carte.



Note : pour débiter, nous vous conseillons nos activités « Flash ».

Faites un clic droit sur le bloc **Arduino - générer le code** puis cliquez sur **Téléverser dans l'Arduino**.



Une nouvelle page s'affiche sur la droite de votre écran. Vous pouvez voir le code de votre programme dans cette nouvelle fenêtre.



Cliquez sur **Téléverser dans l'Arduino**.

Vous pouvez apercevoir la boîte de dialogue qui indique le téléversement en cours. À la fin de celui-ci, cette même boîte de dialogue affichera « Téléversement fini ».



À la fin du téléversement, **IZlone** exécute le programme.

IMPORTANT : Si vous êtes amené à modifier/changer votre programme, il n'est pas nécessaire de téléverser dans l'Arduino. Le code est modifié automatiquement. Lancez directement le téléversement.

Fiches d'activité

Comment ça marche ?

Pour chaque activité :

- nous vous indiquons le(s) extension(s) utilisée(s) et les concepts de programmation abordés
- nous détaillons la réalisation de l'activité avec écrans et commentaires,
- nous proposons un corrigé.

Pour aller plus loin, nous vous proposons également des programmes pour réaliser des jeux comme Tetris, Pong... à vous de jouer !

ACTIVITÉ FLASH : Programmer les capteurs

Extension utilisée : IZlone

Durée : 15 min

Les concepts

Téléversement de programme
Utilisation de fonctions simples
Utilisation Attendre jusqu'à
Utilisation de la boucle infinie
Utilisation de la fonction Si...Alors...

Réalisation 1

Attendre jusqu'à ce que le bouton du haut soit appuyé pour faire apparaître sur l'écran de l'IZlone une flèche rouge vers le haut :

Testez et observez le résultat.

Arduino - générer le code

attendre jusqu'à bouton vers le haut relâché

afficher un symbole ou un smiley ↑ avec la couleur rouge

Réalisation 2

Faire apparaître sur l'écran de la IZlone une flèche vers le bas rouge quand le bouton du haut est appuyé, une flèche vers le haut rouge quand le bouton du bas est appuyé :

Testez et observez le résultat.

Arduino - générer le code

répéter indéfiniment

si bouton vers le haut relâché alors

afficher un symbole ou un smiley ↓ avec la couleur rouge

si bouton vers le bas relâché alors

afficher un symbole ou un smiley ↑ avec la couleur rouge

Réalisation 3

Faire apparaître sur l'écran de la IZlone une flèche vers le bas orange quand le bouton du haut est appuyé, une flèche vers le haut verte quand le bouton du bas est appuyé, le logo IZlproto en vert quand le bouton **Start** est relâché et choisissez ce que vous voulez pour les boutons gauches et droits :

Testez et observez le résultat.

Arduino - générer le code

répéter indéfiniment

si bouton vers le haut relâché alors

afficher un symbole ou un smiley ↓ avec la couleur orange

si bouton vers le bas relâché alors

afficher un symbole ou un smiley ↑ avec la couleur vert

si bouton commencer relâché alors

afficher un symbole ou un smiley (z) avec la couleur vert

ACTIVITÉ FLASH : Programmer une Lettre

Extension utilisée : IZlone

Durée : 15 min

Les concepts

Téléversement de programme
Utilisation de fonctions simples
Mettre en pause un programme
Utilisation de la boucle infinie
Vider l'affichage

Réalisation 1

Faire apparaître sur l'écran de l'IZlone la lettre A en orange :

Testez et observez le résultat.

Arduino - générer le code

afficher le caractère A avec la couleur orange

Réalisation 2

Faire apparaître sur l'écran de l'IZlone la lettre A en orange pendant 4 secondes, puis faire apparaître la lettre Z en vert :

Testez et observez le résultat.

Arduino - générer le code

afficher le caractère A avec la couleur orange

attendre 4 secondes

afficher le caractère Z avec la couleur vert

Réalisation 3

Faire apparaître sur l'écran de la IZlone la lettre A en orange pendant 4 secondes, puis faire apparaître la lettre Z en vert pendant 3 secondes, puis vider l'écran pendant 2 secondes et tout recommencer indéfiniment :

Testez et observez le résultat.

Arduino - générer le code

répéter indéfiniment

afficher le caractère A avec la couleur orange

attendre 4 secondes

afficher le caractère Z avec la couleur vert

attendre 3 secondes

vider l'affichage

attendre 2 secondes

Note finale : Si cette activité vous a plu, nous vous conseillons d'essayer de réaliser un alphabet en entier. Mélangez des lettres majuscules et minuscules entre elles.

ACTIVITÉ FLASH : Programmer un chiffre

Extension utilisée : IZlone

Durée : 15 min

Les concepts

Téléversement de programme
Utilisation de fonctions simples
Mettre en pause un programme
Utilisation de la boucle infinie
Vider l'affichage

Réalisation 1

Faire apparaître sur l'écran de l'IZlone le chiffre 5 en vert :

Testez et observez le résultat.

Arduino - générer le code

afficher le nombre 5 avec la couleur vert

Réalisation 2

Faire apparaître sur l'écran de la IZlone le chiffre 5 en vert, attendre 2 secondes, faire apparaître le chiffre 1 en rouge :

Testez et observez le résultat.

Arduino - générer le code

afficher le nombre 5 avec la couleur vert

attendre 2 secondes

afficher le nombre 1 avec la couleur rouge

Réalisation 3

Faire apparaître sur l'écran de la IZlone le chiffre 5 en vert, attendre 2 secondes, faire apparaître le chiffre 1 en rouge pendant 4 secondes, vider l'affichage pendant 1,5 s et tout recommencer indéfiniment :

Testez et observez le résultat.

Arduino - générer le code

répéter indéfiniment

afficher le nombre 5 avec la couleur vert

attendre 2 secondes

afficher le nombre 1 avec la couleur rouge

attendre 4 secondes

vider l'affichage

attendre 1.5 secondes

Note finale : Si cette activité vous a plu, nous vous conseillons de passer à l'activité sur le « Compte à rebours ».

ACTIVITÉ FLASH : Programmer un Smiley

Extension utilisée : IZlone

Durée : 15 min

Les concepts

Téléversement de programme
Utilisation de fonctions simples
Mettre en pause un programme
Utilisation de la boucle infinie
Vider l'affichage

Réalisation 1

Faire apparaître sur l'écran de l'IZlone un Smiley rouge qui sourit :

Testez et observez le résultat.

Arduino - générer le code

afficher un symbole ou un smiley 😊 avec la couleur rouge

Réalisation 2

Faire apparaître sur l'écran de la IZlone un Smiley rouge qui sourit, attendre 3 secondes, faire apparaître un autre Smiley orange :

Testez et observez le résultat.

Arduino - générer le code

afficher un symbole ou un smiley 😊 avec la couleur rouge

attendre 3 secondes

afficher un symbole ou un smiley 😄 avec la couleur orange

Réalisation 3

Faire apparaître sur l'écran de la IZlone un Smiley rouge qui sourit, attendre 3 secondes, faire apparaître un autre Smiley orange pendant 5 secondes, vider l'écran pendant 2 secondes puis tout recommencer :

Testez et observez le résultat.

Arduino - générer le code

répéter indéfiniment

afficher un symbole ou un smiley 😊 avec la couleur rouge

attendre 3 secondes

afficher un symbole ou un smiley 😄 avec la couleur orange

attendre 5 secondes

vider l'affichage

attendre 2 secondes

ACTIVITÉ : Apprendre à programmer un ascenseur - Séance 1

Extension utilisée : IZlone

Partie 1 - Comprendre

1

Explications

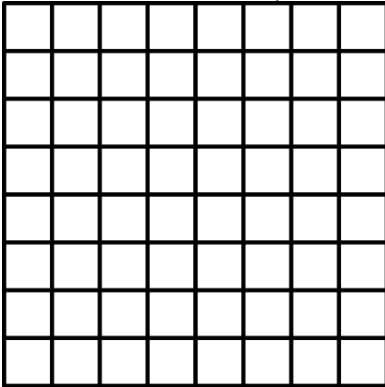
Dans cette activité, tu vas programmer sur ton IZlone un ascenseur qui se déplace entre deux murs. La taille de ton ascenseur est de 3 pixels de largeur et 1 seul de hauteur. Les murs encadrent l'ascenseur et ont une hauteur de 8 avec une épaisseur de 1 pixel.

2

Restitution

ESSAIE

À partir de l'explication ci-dessus, réalise à la main un dessin de l'exercice de l'ascenseur. Restitue fidèlement l'ascenseur puis les murs dans le cadre suivant :



Conseil : pour bien distinguer les murs de l'ascenseur, utilise deux couleurs différentes.

Partie 2 – Les murs

1

Blocs murs

Il va falloir faire afficher sur l'écran de ton IZlone tes deux murs qui encadrent l'ascenseur. Pour cela, il faut faire comprendre à la machine que l'on veut deux blocs de 8 pixels de long qui s'allument et qui ne bougent pas.

Attention, l'origine (1,1) de la matrice est le pixel tout en haut à gauche.

DÉFINIS

Quelles sont les 2 colonnes que tu as choisies pour réaliser tes murs? Cela fait intervenir les notions d'abscisse et d'ordonnée. Qui varie et qui reste fixe?

.....
.....
.....

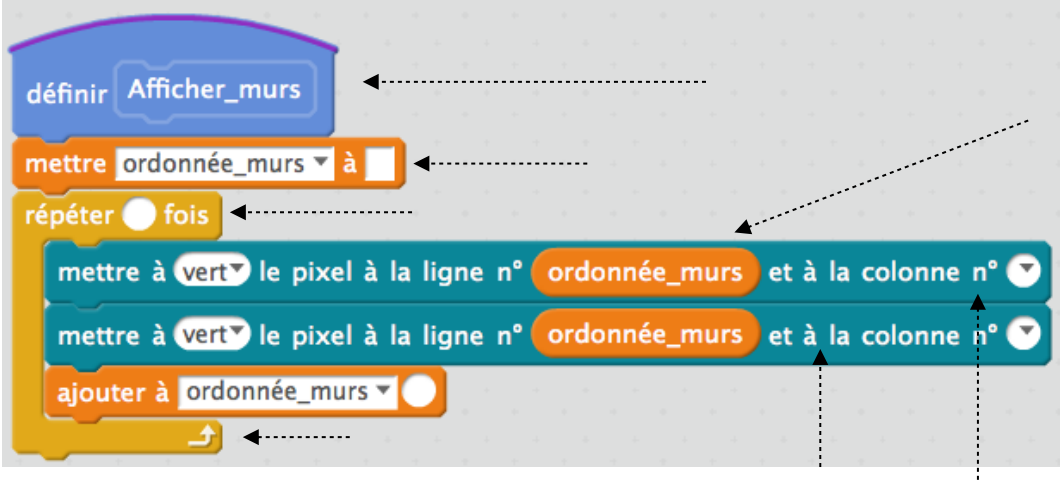
2

Afficher les murs

COMPLÈTE

Remplis le morceau de programme qui correspond à l’affichage des murs et explique ce que représente chaque bloc au bout des flèches.

IMPORTANT : Par convention l’origine (1,1) de la matrice d’IZlone est le pixel tout en haut à gauche de l’écran.



Conseil : dans le choix de l’emplacement de tes deux murs, choisis un écart de 3 pixels entre tes colonnes afin d’y glisser ton ascenseur.

EXPLIQUE

Explique ce que le programme fait ligne par ligne.

.....

.....

.....

.....

.....

.....

.....

Partie 3 – L’ascenseur

1

Bloc ascenseur

Tu as programmé les murs qui encadrent ton ascenseur, il faut à présent programmer ce pour quoi tu es là... C’est-à-dire l’ascenseur en lui-même.

Poursuis

Écris l’ensemble des groupes de coordonnées où l’ascenseur peut aller. Par exemple, lorsqu’il est au rez-de-chaussée, l’ascenseur sera à la ligne 8 et aux coordonnées suivantes :

Groupe 8 (4,8), (5,8), (6,8).

Complète le reste des coordonnées ci-dessous.

.....

.....

.....

.....

2

Afficher Ascenseur

CONTINUE

Maintenant, il faut que tu fasses apparaître l'ascenseur qui va évoluer entre les murs. Sur le même principe que les murs, complète le programme afin d'obtenir un bloc de 3 pixels.

```
définir Afficher_ascenseur
Afficher_murs
mettre abscisse à 
mettre à rouge le pixel à la ligne n° hauteur et à la colonne n° abscisse
ajouter à abscisse 
mettre à rouge le pixel à la ligne n° hauteur et à la colonne n° abscisse
ajouter à abscisse 
mettre à rouge le pixel à la ligne n° hauteur et à la colonne n° abscisse
```

PUIS

explique ce qui se passe ligne par ligne. Explique surtout pourquoi dans le morceau de programme ci-dessus, il y a la variable « hauteur » et pas la variable « ordonnée ».

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Partie 4 – Réalisation

1

C'est l'heure de programmer sur mBlock

VA SUR MBLOCK

Lance mBlock et installe l'extension IZlone. Souvent l'extension est déjà installée sur ton poste.

LANCE-TOI

Réalise les morceaux de programme sur lesquels tu as travaillé jusque-là. Pour le moment, tu ne peux pas encore voir ton travail sur ton IZlone. Ce que tu viens de faire est un travail préparatoire pour la prochaine séance.

2

Conserve ton programme

SAUVEGARDE

Sauvegarde sur ta session tes morceaux de programme. Ils te seront utiles pour la séance 2 où tu programmeras les premiers déplacements de l'ascenseur sur ton IZlone.

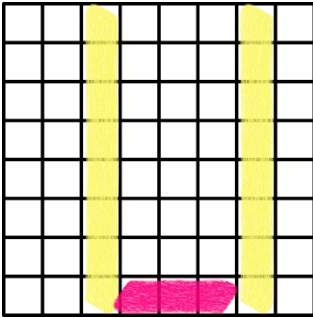
ACTIVITÉ : Apprendre à programmer un ascenseur - Séance 1 - Corrigé

Partie 1 - Comprendre

2

Restitution

Nous vous proposons le dessin suivant où nous avons centré les choses. Vous êtes libre de la disposition de vos éléments. Cependant, nous vous conseillons de suivre notre corrigé afin d'éviter tout problème.



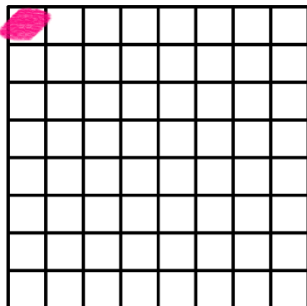
En jaune : les murs / En violet : l'ascenseur
Le tout aux dimensions citées dans l'énoncé et de 2 couleurs différentes.

Partie 2 – Les murs

1

Blocs murs

Dans cette partie, il est très important de préciser que pour IZlone, l'origine de la matrice à LED, c'est-à-dire l'écran, est située en haut à gauche. C'est l'origine (1,1) comme montrée ci-dessous.



En violet, vous avez l'origine de la matrice aux coordonnées (1,1)

Dans la question, il est demandé de choisir deux colonnes, comme dans le dessin dans la Partie 1 pour réaliser vos murs dans votre programme. Vous pouvez tout à fait conserver les coordonnées de vos murs déjà sélectionnées.

C'est ce que nous ferons ici.

C'est pourquoi nous avons choisi la colonne 3 et la colonne 7 pour réaliser nos murs.

Pour dessiner les murs de votre ascenseur avec votre IZlone, vous devez les programmer sur mBlock. Vous allez devoir indiquer à votre IZlone les coordonnées de vos murs. Mais inutile de les renseigner pixel par pixel. Vous allez demander à la machine de le faire pour vous (mais nous verrons ces éléments un peu plus tard dans l'activité).

C'est pourquoi les notions d'abscisse et d'ordonnée sont importantes.
Dans notre cas, les abscisses déterminées pour notre Mur1, $x=3$, et pour notre Mur2, $x=7$, vont rester fixes.

En revanche les ordonnées vont varier « vers le bas » afin de dessiner le reste du mur. En effet nos murs vont se construire du haut vers le bas puisque nous allons fixer notre première ordonnée à la ligne 1. Pas de panique, on vous explique tout dans la question suivante.

2

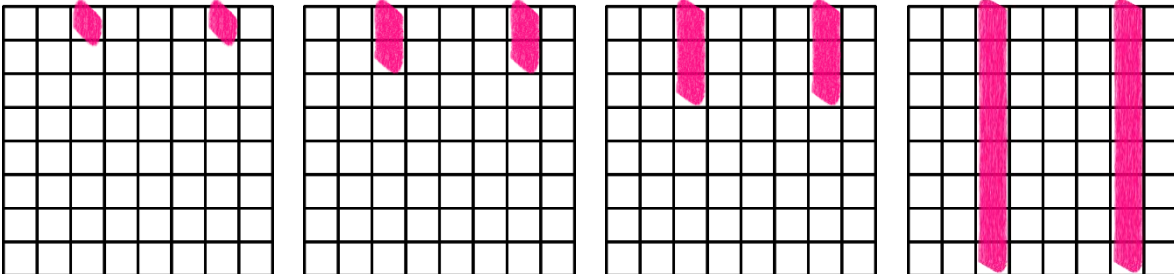
Afficher les murs

Voici le morceau de programme corrigé :

```
defini Afficher_murs
mettre ordonnée_murs à 1
répéter 8 fois
mettre à vert le pixel à la ligne n° ordonnée_murs et à la colonne n° 3
mettre à vert le pixel à la ligne n° ordonnée_murs et à la colonne n° 7
ajouter à ordonnée_murs 1
```

Ici nous mettons l'origine des murs sur la ligne 1
On répète 8 fois l'opération suivante :
Allumer sur la ligne 1 les pixels aux colonnes 3 et 7.
Puis ajouter 1 à l'ordonnée pour passer à la ligne 2.

De cette manière les murs se construisent de la façon suivante :



Jusqu'à la fin de la fonction répéter

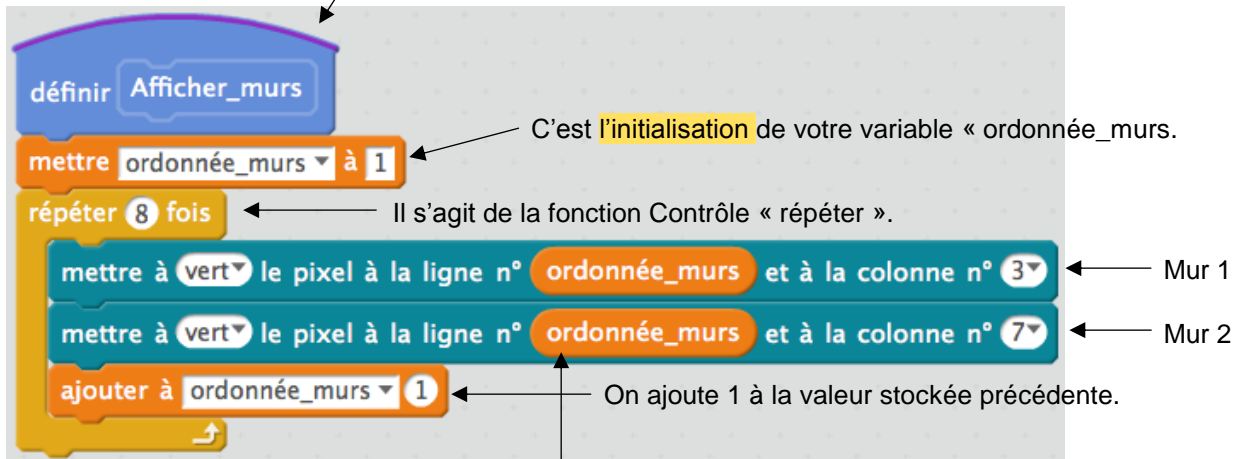
Ainsi vous venez de programmer vos deux murs.

Dans les lignes bleues, la fonction « allumer le pixel » attend des entiers comme arguments (numéro de la ligne et de la colonne). Pourtant nous avons mis une variable pour la ligne. Une variable va nous permettre de stocker une valeur que l'on renseigne dans « mettre ordonnée_murs à 1 ». Ici la variable « ordonnée_murs » = 1.

Donc le programme comprend au moment des « lignes bleues » : Allumer pixel à la ligne 1 et à la colonne 3. On passe par une variable, car on peut facilement les changer grâce à la fonction « ajouter à ordonnée_murs 1 » et donc nous ne sommes pas obligés de redéfinir manuellement chaque coordonnée. La machine calcule pour nous.

Notez que nous aurions pu construire les murs du bas vers le haut en mettant la variable « ordonnée_murs » à 8 et en ajoutant -1 à la variable.
Vous pouvez essayer, le résultat sera le même.

Il s'agit de la fonction Afficher_murs que vous devez créer en allant dans mBlock : Scripts > Blocs & variables > créer un bloc. Une fenêtre de dialogue s'ouvre, tapez le nom de votre fonction et elle apparaîtra dans votre programme.



C'est la variable « ordonnée_mur » qu'il faut créer dans mBlock : Scripts > Blocs & variables > Créer une variable. Une fenêtre de dialogue s'ouvre, renseignez le nom de la variable.

Partie 3 – L'ascenseur

1

Bloc ascenseur

Groupe1 (4 ; 1), (5 ; 1), (6 ; 1)
 Groupe2 (4 ; 2), (5 ; 2), (6 ; 2)
 Groupe3 (4 ; 3), (5 ; 3), (6 ; 3)
 Groupe4 (4 ; 4), (5 ; 4), (6 ; 4)
 Groupe5 (4 ; 5), (5 ; 5), (6 ; 5)
 Groupe6 (4 ; 6), (5 ; 6), (6 ; 6)
 Groupe7 (4 ; 7), (5 ; 7), (6 ; 7)
 Groupe8 (4 ; 8), (5 ; 8), (6 ; 8)

Bien entendu, comme pour les murs, nous allons demander à la machine de calculer ces coordonnées pour nous. Sinon, ça nous prendrait beaucoup trop de temps.

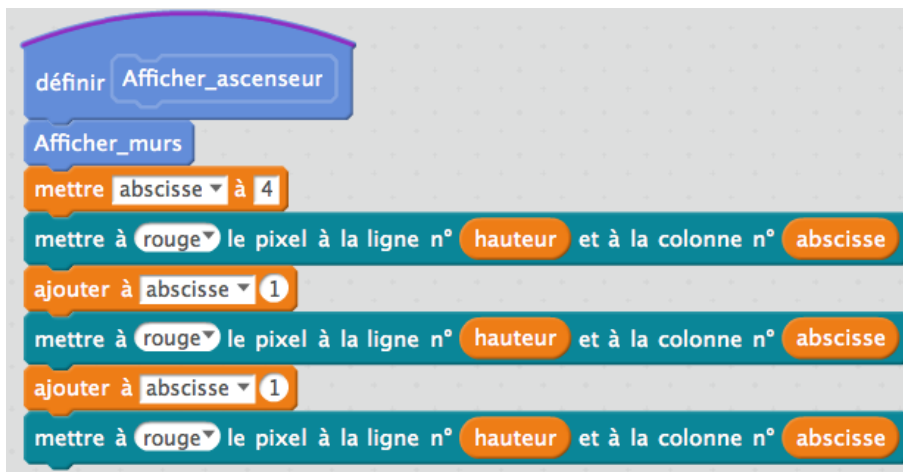
2

Afficher Ascenseur

De la même manière, nous allons définir une fonction « Afficher_ascenseur » pour générer un groupe de trois pixels de largeur afin de représenter l'ascenseur.

Vous allez devoir créer une variable « abscisse » pour situer le point de départ du dessin de votre ascenseur.

Contrairement aux murs, ici l'abscisse varie pour ajouter un pixel à allumer jusqu'à atteindre un groupe de 3 de largeur.



Ici nous n'avons pas utilisé la fonction « répéter ». Nous avons dessiné l'ascenseur manuellement. Il est cependant tout à fait possible de le faire automatiquement.

En revanche la hauteur de l'ascenseur varie. Elle n'est pas fixe. Elle sera définie par une autre variable, ici « hauteur ». L'ordonnée de l'ascenseur va devoir évoluer.

En l'état, vous avez un ascenseur dont le premier pixel est situé sur la colonne n° 4 suivi de 2 autres pixels. Un à la colonne 5 et l'autre à la colonne 6. En effet la machine place le « curseur » de l'ascenseur à la colonne 4 (« mettre abscisse à 4 »), allumera le pixel correspond lorsque toutes les variables auront une valeur, puis décale le « curseur » vers la colonne 5 (« ajouter à abscisse 1 ») etc...

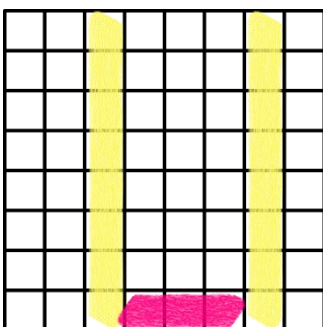
Cependant, la hauteur n'est pas définie, il faut faire preuve d'abstraction pour se représenter la chose. Votre ascenseur peut être à n'importe quelle valeur de la variable « hauteur ». Cependant, nous n'avons pas attribué de valeur à cette dernière.

Ce sera tout l'enjeu de la séance 2.

Nous avons préféré le terme de « hauteur » plutôt qu'« ordonnée », car c'est plus parlant et plus visuel. Un programme a besoin de pouvoir être lu par d'autres programmeurs et lorsqu'on évoque un ascenseur, on se représente immédiatement ce qu'est la hauteur.

Dans ce programme, nous créons et définissons la fonction ascenseur,
 Nous demandons à afficher la fonction « afficher_murs » que nous avons réalisée précédemment,
 On définit le point de départ de notre ascenseur en stockant la valeur 4 dans la variable « abscisse ».
 On allume le pixel à la ligne « hauteur » et la colonne 4.
 On ajoute 1 à la valeur de la variable « abscisse » qui passe donc à 5.
 On allume le pixel à la ligne « hauteur » et à la colonne 5.
 On ajoute 1 à la valeur de la variable « abscisse » qui passe donc à 6.
 On allume le pixel à la ligne « hauteur » et à la colonne 6.

Pour vous aider à vous représenter l'ascenseur, il suffirait de rajouter dans le programme :
 « Mettre "hauteur" à 8 » pour que votre ascenseur soit tout en bas de votre écran, ligne 8 occupant 3 pixels colonnes 4,5 et 6. Vous obtiendriez ceci :



Extension utilisée : IZlone

Partie 1 - Comprendre

1

Explications

Dans cette séance 2, l'objectif va être de pouvoir déplacer l'ascenseur tout entier ligne par ligne entre les murs et sans sortir de l'écran.

2

Restitution

À TOI !

Dans quels sens peut aller l'ascenseur ? Peut-il bouger de gauche à droite, pourquoi ?

.....
.....
.....

CONTINUE

À l'aide de quels actionneurs vas-tu pouvoir faire bouger l'ascenseur, combien y a-t-il d'actionneurs utiles ?

.....
.....
.....

Astuce : N'oublie pas le bouton **Start** pour commencer votre programme.

DÉCRIS

Que se passe-t-il lorsque tu utilises les actionneurs ? Décris pour chaque actionneur ce qu'il se passe.

.....
.....
.....
.....

Maintenant recommence ta description en utilisant la formulation suivante : Si ... alors ...

.....
.....
.....

De quel type de fonction s'agit-il ? (Si...Alors...)

.....
.....
.....

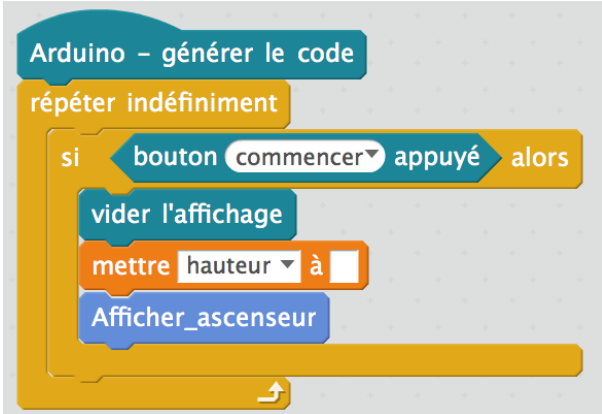
Partie 2 – L'initialisation

1

Afficher l'ascenseur

Souviens-toi de la fin de la séance 1, tu as programmé l'affichage des murs et de l'ascenseur sans pouvoir les voir. En effet, tu n'avais pas donné de valeur à la variable « hauteur ». C'est le moment de le faire.

COMPLÈTE



Voici l'initialisation du programme de l'ascenseur, complète-le pour faire en sorte d'afficher l'ascenseur en bas de l'écran. Donne une valeur à la variable « hauteur ».

Pour rappel, l'origine de la matrice (1,1) est située en haut à gauche du pavé à LED.

EXPLIQUE

Décris ce qui se passe dans le programme ligne par ligne et explique la fonction de chaque bloc.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

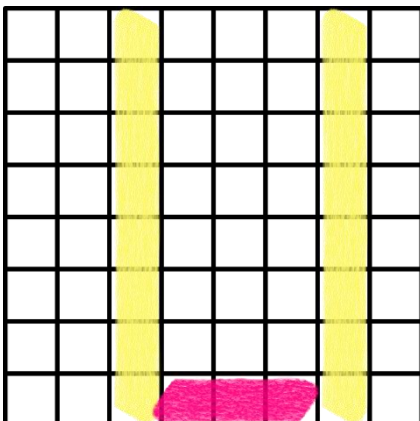
2

Va sur mBlock

Retrouve ta précédente session

Reproduis dans mBlock et sur le fichier que tu as réalisé lors de la séance 1 le morceau de programme ci-dessus.

Génère le code et vérifie que ton ascenseur ainsi que tes murs sont bien présents au bon endroit.



Si tu obtiens sur ton IZlone un résultat semblable à celui-là, tu as bien construit ton programme.

Les murs doivent apparaître aux coordonnées que tu as **fixées** lors de la séance 1.

Ton ascenseur doit quant à lui être positionné entre les murs et à la position que tu lui as donnée grâce à la variable « hauteur ».

Tu peux t'amuser à changer la valeur de la variable « hauteur » dans ton programme pour modifier la position de ton ascenseur.

Tu pourras alors constater qu'il faut à chaque fois générer le code et que tu perds du temps. On va remédier à ça...

PUIS

Explique pourquoi il y a la fonction “et” dans le programme ci-dessus. Comment le programme comprend cette fonction ?

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

3

Vers le bas maintenant

FINIS

```
Arduino - générer le code
répéter indéfiniment
  si bouton commencer appuyé alors
    vider l'affichage
    mettre hauteur à 8
    Afficher ascenseur
  si alors
  si alors
```

Termine le programme dans mBlock, pour que l'ascenseur puisse aller vers le bas également. Pense à ne pas le faire sortir de l'écran. La structure finale doit ressembler à ça. Remplis le reste.

Enregistre ton travail dans ta session. Tu en auras besoin pour la séance3 !

ACTIVITÉ : Apprendre à programmer un ascenseur - Séance 2 - Corrigé

Partie 1 - Comprendre

2

Restitution

Dans quels sens peut aller l'ascenseur ? Peut-il bouger de gauche à droite, pourquoi ?

La réponse attendue est que l'ascenseur peut aller uniquement vers le haut ou vers le bas. Comme dans la réalité. L'ascenseur ne peut pas se déplacer latéralement, car nous l'avons programmé ainsi lors de la séance1.

Souvenez-vous, c'était la dernière question de l'activité. Nous avons programmé l'ascenseur pour évoluer uniquement dans les colonnes 4,5 et 6 de votre IZlone. L'ascenseur ne peut varier que sur l'axe des ordonnées, c'est-à-dire évoluer sur les différentes lignes de votre IZlone.

À l'aide de quels actionneurs vas-tu pouvoir faire bouger l'ascenseur, combien y a-t-il d'actionneurs utiles ?

Les actionneurs sont les boutons de votre IZlone. Au total vous en avez 5. Mais uniquement 3 seront utiles pour l'exercice.

Que se passe-t-il lorsque tu utilises les actionneurs ? Décris pour chaque actionneur ce qui se passe.

Lorsque vous utilisez un des actionneurs, un bouton, vous envoyez une information à votre IZlone. La machine reçoit un signal et c'est à vous de programmer la façon dont votre IZlone va interpréter ce signal et y répondre.

Lorsque j'appuie sur le bouton du haut une fois, mon ascenseur montera d'une ligne.

Inversement pour le bouton du bas.

Lorsque j'appuie sur le bouton Start, mon programme démarre par exemple.

Maintenant recommence ta description en utilisant la formulation suivante : Si... alors...

Si j'appuie sur le bouton du haut, ALORS l'ascenseur monte

Si j'appuie sur le bouton du bas, ALORS l'ascenseur descend

Si j'appuie sur le bouton Start, ALORS je mets la « hauteur » à 8.

Pourquoi mettre la « hauteur » à 8 ? Souvenez-vous du corrigé de la séance1 de cette activité. Pour vous aider à vous représenter l'ascenseur, nous vous avons proposé de définir la variable « hauteur » (qui définit où l'ascenseur doit être sur l'axe des ordonnées) à 8 pour que l'ascenseur apparaisse en bas de l'écran. C'est un très bon moyen de démarrer votre programme avec votre ascenseur en position « en bas » comme dans la réalité.

Vous constaterez plus tard dans la séance qu'après avoir téléversé votre code dans l'Arduino, le logo « Z » de IZlproto apparaît. Pressez Start et démarrer l'ascenseur en bas est un bon moyen de débiter votre programme.

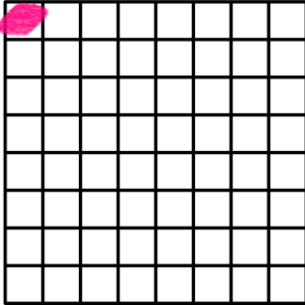
De quel type de fonction s'agit-il ?

Il s'agit d'une fonction conditionnelle très utile. Elle permet à IZlone d'exécuter des actions uniquement sous certaines conditions. Si les conditions initiales sont vraies, alors le programme s'exécute.

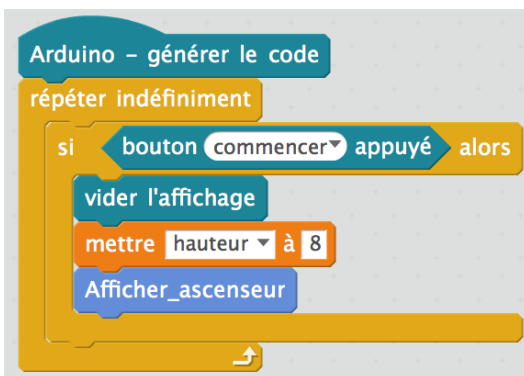
Partie 2 – L'initialisation

1

Afficher l'ascenseur



Tout d'abord nous vous rappelons que l'origine de la matrice (1,1) est le point violet ci-contre.



Voici le programme complété et correct pour initialiser l'ascenseur ligne 8.

La hauteur a bien été mise à 8 pour que l'ascenseur apparaisse à la 8ème ligne et aux colonnes 4,5 et 6.

Décris ce qui se passe dans le programme ligne par ligne et explique la fonction de chaque bloc

- On demande à mBlock de générer le code de notre programme pour que l'Arduino puisse le comprendre.
- On inscrit l'ensemble de notre programme dans une boucle infinie sinon votre IZlone ne vous afficherait le programme qu'une seule fois et vous n'auriez même pas le temps de le voir s'afficher à l'écran. (Vous pouvez faire le test, enlevez la boucle « répéter indéfiniment » et regardez le résultat, vous ne devriez rien voir. C'est parce que l'Arduino exécute le programme à une très grande vitesse. Elle vous affiche l'ascenseur, mais pendant une très courte période).
- On définit une boucle conditionnelle Si... Alors. Cette boucle nous permet de démarrer le programme. Si j'appuie sur Start Alors...
- On vide l'écran, le logo IZlproto est enlevé et place est faite pour l'ascenseur.
- On définit la variable « hauteur » à 8. Votre IZlone sait exactement où placer l'ascenseur.
- On demande à votre IZlone d'exécuter la fonction « Afficher_ascenseur » que vous avez programmée lors de la séance1. Cela permet d'allumer les pixels programmés dans la fonction et qui sont représentés par l'ascenseur avec les murs.
- La boucle conditionnelle est finie, la machine lit qu'il faut remonter au début de la boucle infinie. Si vous refaites Start, le programme s'exécute à nouveau.

2

Va sur mBlock

À ce moment vous devez avoir le programme suivant affiché dans mBlock :

```

Arduino - générer le code
répéter indéfiniment
  si bouton commencer appuyé alors
    vider l'affichage
    mettre hauteur à 8
    Appeler Afficher_ascenseur

définir Afficher_ascenseur
  Appeler Afficher_murs
  mettre abscisse à 4
  mettre à rouge le pixel à la ligne n° hauteur et à la colonne n° abscisse
  ajouter à abscisse 1
  mettre à rouge le pixel à la ligne n° hauteur et à la colonne n° abscisse
  ajouter à abscisse 1
  mettre à rouge le pixel à la ligne n° hauteur et à la colonne n° abscisse

définir Afficher_murs
  mettre ordonnée_murs à 1
  répéter 8 fois
    mettre à vert le pixel à la ligne n° ordonnée_murs et à la colonne n° 3
    mettre à vert le pixel à la ligne n° ordonnée_murs et à la colonne n° 7
    ajouter à ordonnée_murs 1
  
```

Partie 3 – Mouvements

1

Vers le haut...

```

si bouton vers le haut appuyé et hauteur > 1 alors
  vider l'affichage
  ajouter à hauteur -1
  Appeler Afficher_ascenseur
  
```

Avec les morceaux de programme, vous devez essayer de produire le programme ci-contre :

Le but n'est pas que vous trouviez le résultat tout de suite. Mais que vous réfléchissiez à la démarche pour y parvenir.

L'enjeu ici est de parvenir à poser comme base la fonction conditionnelle suivante : Si j'appuie sur le bouton du haut ALORS je dois ajouter une valeur à la variable « hauteur » pour faire évoluer mon ascenseur vers le haut.

La **fonction Si... Alors... Sinon** permet d'exécuter des instructions très utiles. Si les conditions sont vraies, alors j'exécute des instructions, si les conditions sont fausses (sinon) j'exécute d'autres instructions. Ici, il y a **2** conditions : « Appuyé vers le haut » (1) et « hauteur > 1 » (2). Dans notre cas ici, Si « **bouton vers le haut appuyé** » et « **hauteur > 1** » sont **vrais** alors votre IZlone vide l'affichage, retranche 1 à la variable « hauteur » et affiche l'ascenseur.

Si l'une des conditions est fausse, voire les deux, Alors la machine n'exécute plus le programme souhaité : rien. La machine ne fait rien. La fonction « **et** » est très importante ici. Il faut que les 2 conditions soient vraies (et pas une des deux) pour exécuter la partie Alors...

Pourquoi la condition « hauteur > 1 » est ici présente ?

Sans cette condition l'ascenseur ne s'arrêterait pas à la ligne 1 (une fois en haut de l'écran) et disparaîtrait. Votre IZlone exécuterait le programme comme il lui est donné. Je continue de retrancher 1 à la variable « hauteur ».

Mais avec cette condition, dès que l'ascenseur atteint la ligne 1 (le haut de l'écran), la position de l'ascenseur n'est plus supérieure à 1, donc la condition 2 n'est plus vraie.

Souvenez-vous que pour faire monter l'ascenseur il faut que les conditions 1 et 2 soient vraies.

Là ce n'est plus le cas. La condition 1 est vraie (j'appuie sur le bouton), mais plus la 2. D'où l'intérêt de la fonction « Et ». Avec la fonction « Ou » par exemple, tout changerait. Il faut uniquement que l'une des deux conditions soit vraie.

En outre pourquoi devoir ajouter à « hauteur » -1 ?

C'est très simple. Souvenez-vous que votre ascenseur est à la ligne 8. En effet, vous avez demandé dans l'initialisation de mettre « hauteur » à 8. Pour la machine « hauteur » = 8. Il faut faire diminuer cette valeur pour que la machine fasse remonter l'ascenseur. Cependant la variable ne comprend pas l'ordre « remonter ». Elle ne comprend que les chiffres. Donc, il faut retrancher à 8 la valeur 1 pour que la variable stocke une nouvelle valeur : 7. Dès lors, la variable « hauteur » est égale à 7. Dans votre fonction « Afficher_ascenseur », la variable « hauteur » définit la position de votre ascenseur sur l'axe des ordonnées. Si « hauteur » =7 alors votre IZlone affiche votre ascenseur à la ligne 7 et ainsi de suite jusqu'à 1 qui est la limite. Rappelez-vous aussi de l'origine de la matrice (1,1) en haut à gauche de l'écran.

Enfin quant à l'enchaînement d'instructions « vider l'affichage », « ajouter à hauteur -1 » et « Afficher_ascenseur », l'ordre et la fonction des instructions sont très importants.

« Vider l'affichage » permet de supprimer tout ce qu'il y a sur l'écran de votre IZlone. À ce moment le programme exécuté supprime l'ascenseur à la ligne 8 et supprime les murs. L'écran est alors vide.

Le programme exécute l'action suivante : ajouter à « hauteur » -1. Les coordonnées de l'ascenseur sont modifiées. Elles passent à la ligne 7. L'instruction suivante demande à la machine de faire réapparaître l'ascenseur au bon endroit **et** avec les murs. En effet les murs apparaissent aussi à ce moment-là.

Souvenez-vous, lors de la séance1, lors de la création de la fonction « Afficher_ascenseur », la première ligne est « Afficher_murs ». Ainsi dès que vous faites apparaître l'ascenseur, les murs apparaissent également. C'est un moyen d'automatiser l'affichage des murs. Ils sont imbriqués dans la fonction « Afficher_ascenseur ». Sinon ils risqueraient de disparaître lors de la requête « vider l'affichage ». À ce moment-là, votre ascenseur est monté d'une ligne et les murs sont toujours aux bonnes coordonnées.

Vous noterez que la machine exécute cette succession d'instructions à une très grande vitesse. C'est pour ça que vous ne voyez pas tout disparaître puis tout réapparaître.

« Vider l'affichage » permet aussi à l'ascenseur de conserver sa taille de 3 pixels de largeur et 1 d'épaisseur. Sinon le programme ferait apparaître un ascenseur à la ligne 8 **et** à la ligne 7. En effet, la fonction « Afficher_ascenseur » prévoit uniquement d'allumer 3 pixels, pas de les éteindre.

Nous sommes bien conscients de toute la masse d'informations à retenir. Prenez le temps de bien relire chaque partie. Il faut intégrer chaque concept afin de bien comprendre ce code qui semble pourtant si simple.

2

...nous allons

Explique pourquoi dans le programme ci-dessous il est écrit « si hauteur >1 Alors vide l’affichage puis ajoute -1 à hauteur puis affiche l’ascenseur ?

Vous avez déjà les réponses à ces questions dans la partie précédente.

Expliquez pourquoi il y a la fonction “et” dans le programme ci-dessus. Comment le programme comprend cette fonction ?

Idem

3

Vers le bas maintenant

De la même manière que pour faire monter l’ascenseur, vous devez construire le programme pour le faire descendre.

```
Arduino - générer le code
répéter indéfiniment
  si bouton commencer appuyé alors
    vider l’affichage
    mettre hauteur à 8
    Afficher ascenseur
  si bouton vers le haut appuyé et hauteur > 1 alors
    vider l’affichage
    ajouter à hauteur -1
    Afficher ascenseur
  si bouton vers le bas appuyé et hauteur < 8 alors
    vider l’affichage
    ajouter à hauteur 1
    Afficher ascenseur
```

Voici le programme complet qui vous permet de faire monter et descendre votre ascenseur en actionnant les boutons haut et bas de votre IZlone.

Lors de la séance3, nous programmerons un ascenseur un peu plus réaliste. En effet, avec les actionneurs nous automatiserons encore plus l’ascenseur. Il se rendra tout seul à différents étages, comme les vrais...

Extension utilisée : IZlone

Partie 1 - Comprendre

1

Explications

Lors de cette dernière séance de l'activité ascenseur, nous allons programmer un ascenseur pour qu'il se rende automatiquement à différents étages.

Dans les ascenseurs que tu prends au quotidien, tu n'appuies pas sur un bouton vers le haut pour monter d'un étage. Tu appuies sur « étage 3 » et l'ascenseur s'y rend. C'est ce que nous allons faire aujourd'hui.

Si bouton gauche appuyé, l'ascenseur doit aller au 3^{ème} étage (ligne 2 de la matrice).

Si bouton droit appuyé, l'ascenseur doit aller au 1^{er} étage (ligne 6).

Si bouton haut appuyé, l'ascenseur doit aller au RDC (ligne 8).

Si bouton bas appuyé, l'ascenseur doit aller au 2^{ème} étage (ligne 4).

2

Restitution

SOUVIENS-TOI !

Quelle va être la différence majeure avec ce que tu as fait lors de la séance2?

.....
.....
.....
.....

CONTINUE

À quoi servent les 4 dernières lignes de l'énoncé ?

.....
.....
.....
.....

DÉCRIS

Comment ferais-tu dans mBlock pour programmer les 4 dernières lignes de l'énoncé pour automatiser ton ascenseur ?

.....
.....
.....
.....

3

Va sur mBlock

ESSAIE

Tente de programmer ton ascenseur automatique selon tes idées. Prends 5 min pour essayer.

ATTENTION : Réutilise les fonctions « Afficher_ascenseur » et « Afficher_murs » que tu as programmées lors de la séance1.

Partie 2 - Consigne

Nous allons te proposer notre façon de faire un ascenseur automatique.

1

Créer une fonction « Aller_à_consigne »



Récrée ce bloc sur mBlock.

2

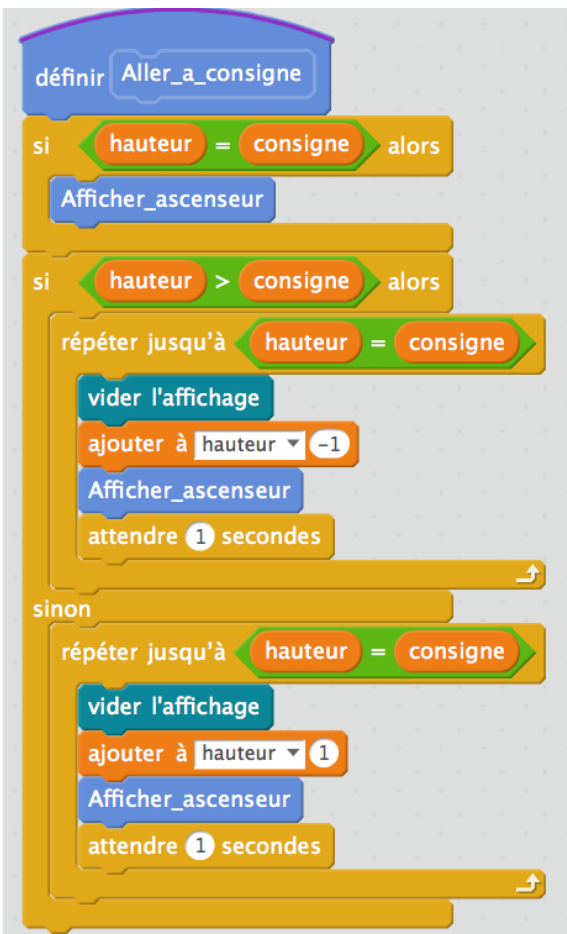
Créer une variable « consigne »



Recrée cette variable sur bloc également.

3

Explication de la fonction consigne



EXPLIQUE

Quel est le principe de cette fonction « Aller_à_consigne » ?

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Aide : La consigne représente l'étage que tu désires atteindre, et tu dois considérer cette fonction comme un comparateur.

KEEP GOING

À quoi sert le bloc « attendre 1 seconde » ? Que se passera-t-il si je modifie le temps à attendre ?

.....
.....
.....

IT IS NOT DONE

Rappelle l'utilité de la fonction SI...ALORS...SINON.

.....
.....

À quoi sert la fonction « répéter jusqu'à » ?

.....
.....
.....

4

Va sur mBlock

CONTINUE

Va sur mBlock et reproduis la fonction « aller_à_consigne »

Partie 3 - Automatisation

1

Définition des étages

VAS-Y

Reproduis le programme ci-contre dans mBlock puis modifie le contenu des blocs des boutons à appuyer.

Ils sont tous sur « commencer ».

Enfin, remplis les cases manquantes des consignes pour qu'elles correspondent aux étages dans l'énoncé.

```

Arduino - générer le code
répéter indéfiniment
si bouton commencer appuyé alors
  vider l'affichage
  mettre hauteur à 8
  Afficher ascenseur
si bouton commencer appuyé alors
  mettre consigne à 
  Aller_a_consigne
si bouton commencer appuyé alors
  mettre consigne à 
  Aller_a_consigne
si bouton commencer appuyé alors
  mettre consigne à 
  Aller_a_consigne
si bouton commencer appuyé alors
  mettre consigne à 
  Aller_a_consigne

```

2

Comment ça marche ?

```

si bouton à droite appuyé alors
  mettre consigne à 6
  Aller_a_consigne

```

BIENTÔT FINI

Explique le fonctionnement du morceau de programme ci-dessus.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

3

C'est l'heure de tester

À ce stade, si tu as bien suivi toutes les instructions et bien rempli les différents programmes. Tu devrais avoir fini d'automatiser ton ascenseur.

Il devrait pouvoir se déplacer seul grâce aux instructions que tu lui donnes.



ACTIVITÉ : Apprendre à programmer un ascenseur - Séance 3 - Corrigé

Partie 1 - Comprendre

2

Restitution

Quelle va être la différence majeure avec ce que tu as fait lors de la séance 2?

Lors de la séance 2, vous avez programmé votre ascenseur pour qu'il se déplace vers le haut une fois ou vers le bas une fois lorsque j'appuie une fois sur un bouton. Pendant cette séance l'objectif va être d'élaborer un programme qui déclenche **plusieurs** mouvements bien qu'un seul bouton ait été actionné.

À quoi servent les 4 dernières lignes de l'énoncé ?

Ce sont des instructions qu'il va falloir programmer. Elles vont servir à étalonner votre programme pour que tout le monde ait la même chose.

Comment ferais-tu dans mBlock pour programmer les 4 dernières lignes de l'énoncé pour automatiser ton ascenseur ?

Une des solutions à laquelle on pourrait penser immédiatement serait de programmer la chose suivante : Si j'appuie sur le bouton de gauche, ALORS mettre hauteur à 2. Ainsi l'ascenseur irait bien à la ligne 2, donc à l'étage 3. Cependant, cette instruction « téléporterait » l'ascenseur de la ligne 8 à la ligne 2. Et ça ne représente pas un ascenseur qui monte étage par étage.

Le but ici est de vous faire arriver à cette conclusion : il faut faire autrement, il faut sentir le problème auquel vous serez confronté.

Dans cette activité, nous allons vous proposer une solution pour parvenir à un ascenseur automatisé qui monte étage par étage. Il existe plusieurs façons de le faire bien entendu.

Si toutefois, vous êtes parvenu au résultat souhaité, sans notre aide et sans regarder la suite de la séance, nous vous adressons toutes nos félicitations. Ce n'était pas évident.

Partie 2 - Consigne

1

Créer une fonction « Aller_à_consigne »

Pour créer cette fonction, nous vous rappelons la procédure :

Allez dans l'onglet Scripts > Blocs & variables > Créer un bloc > Une fenêtre de dialogue s'ouvre > indiquez le nom du bloc souhaité > faites OK.

2

Créer une variable « consigne »

Allez dans l'onglet Scripts > Blocs & variables > Créer une variable > une fenêtre de dialogue s'ouvre > indiquez le nom du bloc souhaité > faites OK.

3

Explication de la fonction consigne

Quel est le principe de cette fonction « Aller_à_consigne » ?

Comme mentionné sous l'image de l'activité, c'est un **comparateur**. Cette fonction est là pour comparer la variable « consigne » à la variable « hauteur ».

Mais cette fonction est aussi le **moteur** de notre programme. C'est la partie qui va permettre à l'ascenseur de monter ou de descendre étage par étage.

Comment ça fonctionne ?

Pour bien comprendre le rôle de la fonction, nous devons prendre un peu d'avance sur les questions suivantes. Nous aurons besoin de ceci :



Ce morceau de programme provient du corrigé de cette même séance dans la Partie 3 : Automatisation.

Étape 1 : Considérez que la variable « hauteur » est à 8, comme dans l'initialisation dans la séance 2. L'ascenseur est à la ligne 8 au RDC.

Étape 2 : On presse sur le bouton de gauche pour mettre la consigne à 2. Nous vous rappelons que **la consigne n'est autre que l'étage auquel vous voulez vous rendre**. Enfin on exécute la fonction Aller_à_consigne. (Tout ceci provient des instructions de l'image ci-dessus).



Étape 3 : La fonction Aller_à_consigne exécute son programme.

Étape 4 : La fonction va comparer la variable « consigne » à la variable « hauteur ».

Nous vous rappelons que pour le moment Consigne = 2 et Hauteur = 8 et nous voulons aller au 3ème étage (ligne 2 = 3ème étage)

Étape 4.1 : La fonction « Aller_à_consigne » est en 3 parties. **Une partie** avec la fonction conditionnelle Si... Alors... et **deux autres parties** dans la fonction conditionnelle Si... Alors... Sinon.

Étape 4.2 : Ici nous sommes dans le cas où la hauteur est supérieure à la consigne. On va donc aller dans la deuxième partie du programme (encadrée en rouge ci-contre) Le programme va exécuter les instructions encadrées puisque la condition initiale est vraie : « hauteur » > « consigne ».

Alors, le programme va répéter les instructions suivantes jusqu'à ce que « hauteur » = « consigne » : Il va alors vider tout l'affichage, ajouter -1 à « hauteur » (l'ascenseur passe à la ligne 7 et monte, il n'est pas encore allumé), afficher l'ascenseur (les pixels s'allument), attendre 1 seconde. Ce programme va s'exécuter jusqu'à ce que « hauteur » = « consigne », jusqu'à ce que la hauteur soit égale à 2.

Prenez le temps de bien relire toutes ces étapes pour bien comprendre les mécanismes en place.

Étape 5 Variante : Imaginons le cas suivant. La variable « hauteur » = 2 et la variable « consigne » = 2 (on clique à nouveau sur le bouton de gauche, bien qu'on soit déjà au 3ème étage). Dans ce cas nous serions dans la 1ère partie de la fonction « Aller_à_consigne ». En effet si la condition « hauteur » est égale à « consigne ». Alors le programme ne fait qu'afficher l'ascenseur à sa dernière hauteur connue : ligne 2.

Étape 5.1 Variante : Imaginons le cas suivant. La variable « hauteur » = 2 et la variable « consigne » = 8 (on souhaite se rendre désormais au RDC).

Dans ce cas nous serions dans la 3ème partie de la fonction « Aller_à_consigne ». En effet la fonction a comparé les deux variables. Elle regarde d'abord si elles sont égales : Non (partie 1). Puis elle regarde si « hauteur » est supérieure à « consigne » : Non (partie 2). Cela signifie que la condition initiale est fausse. Donc **nécessairement**, la variable « consigne » est supérieure à la variable « hauteur » puisque tous les autres cas ont été écartés.

Vous remarquerez l'utilité de la fonction Si... Alors... Sinon dans ce cas-là. Elle nous évite de recréer une fonction Si... Alors où l'on y indique la condition initiale « Si "consigne" > "hauteur" Alors fais descendre l'ascenseur ». On utiliserait 2 fonctions, bien qu'on puisse le faire en une. C'est moins rapide et moins efficace.

Le fait que « consigne » > « hauteur » n'est pas présent dans la fonction Si... Alors... Sinon. Pourtant c'est le seul cas possible restant. Et c'est la partie Sinon qui gère l'exécution du programme quand la condition initiale est fausse.

Donc dans cette troisième partie, la fonction « Aller_à_consigne » fait descendre l'ascenseur vers le RDC jusqu'à ce que « hauteur » = « consigne ».

À quoi sert le bloc « attendre 1 seconde » ? Que se passera-t-il si je modifie le temps à attendre ?

C'est le bloc qui va gérer la vitesse à laquelle l'ascenseur va se déplacer. Sans ce bloc, le programme modifierait la variable « hauteur » jusqu'à ce qu'elle atteigne la « consigne » voulue, mais à très grande vitesse. Vous auriez l'impression que l'ascenseur se téléporte. Pour éviter ce problème, on impose au programme d'attendre une seconde avant de modifier la « hauteur ». Pour pouvez essayer de supprimer ce bloc pour vérifier ce qu'il se passe.

En modifiant la variable « seconde », vous allez modifier la vitesse d'ascension et de descente de l'ascenseur. Vous pouvez jouer avec cette variable. Testez et trouvez la vitesse qui vous semble la plus agréable.

Rappelle l'utilité de la fonction SI... ALORS... SINON.

Nous avons déjà répondu à cette question dans la Partie 2 : Consigne point 3

À quoi sert la fonction « répéter jusqu'à » ?

Cette fonction permet de répéter une action tant que la condition initiale n'est pas vraie.

Partie 3 - Automatisation

1

Définition des étages

Voici le corrigé avec les boutons et les consignes qui correspondent à l'énoncé de la séance.

```
Arduino - générer le code
répéter indéfiniment
  si bouton commencer appuyé alors
    vider l'affichage
    mettre hauteur à 8
    Afficher_ascenseur
  si bouton à gauche appuyé alors
    mettre consigne à 2
    Aller_a_consigne
  si bouton à droite appuyé alors
    mettre consigne à 6
    Aller_a_consigne
  si bouton vers le bas appuyé alors
    mettre consigne à 4
    Aller_a_consigne
  si bouton vers le haut appuyé alors
    mettre consigne à 8
    Aller_a_consigne
```

2

Comment ça marche ?

Nous avons déjà répondu à cette question dans la Partie 2 : Consigne point 3. Vous noterez que dans cette question, le bouton droit est appuyé et la consigne est à 6. Ce que nous avons décrit plus haut dans ce corrigé marche exactement de la même façon.

3

C'est l'heure de tester

Voici le programme complet qui vous permet de réaliser un ascenseur automatisé.

```

Arduino - générer le code
répéter indéfiniment
  répéter indéfiniment
    si bouton commencer appuyé alors
      vider l'affichage
      mettre hauteur à 8
      Afficher_ascenseur
    si bouton à gauche appuyé alors
      mettre consigne à 2
      Aller_a_consigne
    si bouton à droite appuyé alors
      mettre consigne à 6
      Aller_a_consigne
    si bouton vers le bas appuyé alors
      mettre consigne à 4
      Aller_a_consigne
    si bouton vers le haut appuyé alors
      mettre consigne à 8
      Aller_a_consigne
  fin
fin

définir Aller_a_consigne
si hauteur = consigne alors
  Afficher_ascenseur
si hauteur > consigne alors
  répéter jusqu'à hauteur = consigne
    vider l'affichage
    ajouter à hauteur -1
    Afficher_ascenseur
    attendre 1 secondes
sinon
  répéter jusqu'à hauteur = consigne
    vider l'affichage
    ajouter à hauteur 1
    Afficher_ascenseur
    attendre 1 secondes
fin

définir Afficher_ascenseur
Afficher_murs
mettre abscisse à 4
mettre à rouge le pixel à la ligne n° hauteur et à la colonne n° abscisse
ajouter à abscisse 1
mettre à rouge le pixel à la ligne n° hauteur et à la colonne n° abscisse
ajouter à abscisse 1
mettre à rouge le pixel à la ligne n° hauteur et à la colonne n° abscisse

définir Afficher_murs
mettre ordonnée_murs à 1
répéter 8 fois
  mettre à vert le pixel à la ligne n° ordonnée_murs et à la colonne n° 3
  mettre à vert le pixel à la ligne n° ordonnée_murs et à la colonne n° 7
  ajouter à ordonnée_murs 1
fin
  
```

C'est la fin de cette activité Ascenseur.

ACTIVITÉ : Apprendre à programmer un compte à rebours

Extension utilisée : IZlone

Partie 1 - Comprendre

1

Explications

Un compte à rebours est un décompte vers une fin connue. C'est une suite de nombres défilant dans l'ordre décroissant. Durant cette activité, nous allons programmer de 2 façons différentes un compte à rebours sur votre IZlone.

2

Restitution

COMMENCE

Que permet de mesurer un compte à rebours et pourquoi est-ce un outil très utile ?

.....
.....
.....
.....

SOUVIENS-TOI

Dans quelles situations utilise-t-on un compte à rebours ? Trouve au moins 3 situations différentes.

.....
.....
.....
.....
.....

Selon toi, quels sont les paramètres importants pour qu'un compte à rebours fonctionne correctement ?

.....
.....
.....
.....
.....

Partie 2 – 1^{ère} méthode, manuellement

1

Parvenir à afficher un nombre sur l'écran de l'IZlone

Pour le compte à rebours que tu t'apprêtes à réaliser, nous te proposons de démarrer à 10 et de finir à 0. Pour cela et dans un premier temps, essaye de faire afficher le nombre 10 sur l'écran de ton IZlone.

VA SUR MBLOCK

Fais afficher un 10 sur ton IZlone. Une fois que tu as réussi, retranscris ton programme dans le cadre ci-dessous.



Aide : tu auras besoin de l'extension « IZlone » pour réaliser cet exercice et de ce bloc :

afficher le nombre ▼

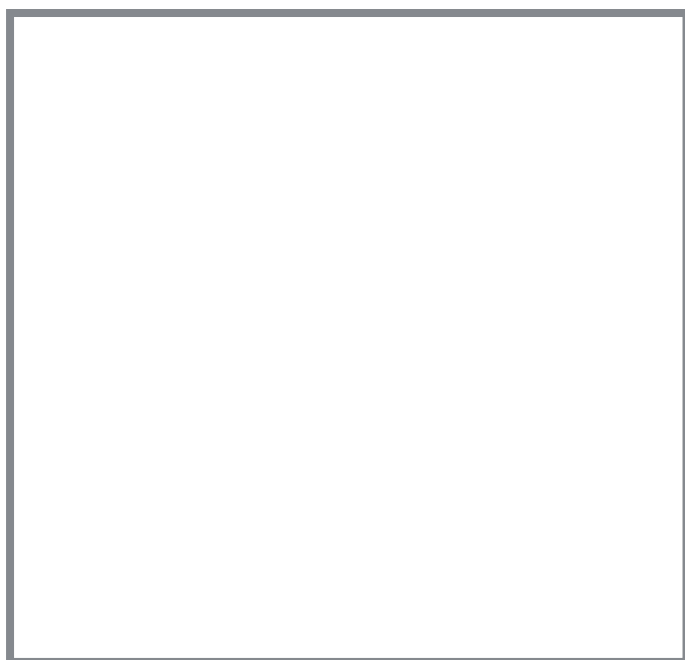
Une fois que tu as réussi, tu peux faire afficher d'autres nombres sur ton IZlone, tu constates que pour afficher un autre nombre, il faut à chaque fois re-téléverser un nouveau programme dans IZlone. On va y remédier.

2

Parvenir à faire afficher une suite de nombre à l'écran.

COMPLÈTE

Il faut que tu crées un programme plus complet afin de faire afficher une suite de nombre sur ton IZlone. Utilise ton travail précédent pour aller plus vite. Utilise la fonction « Dupliquer » de mBlock. N'oublie pas qu'il faut faire un compte à rebours qui va de 10 à 0.



Retranscris ton programme ici dès qu'il fonctionne sur mBlock.

N'oublie pas non plus qu'un compte à rebours mesure le temps qui s'écoule. Règle la vitesse de défilement des nombres sur 1 **seconde**.

Partie 3 – 2^{ème} méthode, automatiquement

1

Faire compter la machine

Tu as pu constater que dans la partie 2 de cette activité, ton compte à rebours représente un programme plutôt long et c'était à toi de renseigner les valeurs. Maintenant, on va faire calculer ton IZlone à ta place. C'est beaucoup moins fatigant.

POURSUIS

Rappelle ou explique ce qu'est une variable en programmation.

.....
.....
.....

CONTINUE

Rappelle ou explique le rôle de la fonction « répéter x fois ».

.....
.....
.....

2

Créer une variable

RETOURNE SUR MBLOCK

A rectangular button with rounded corners, a grey border, and a light orange background. The word "Compteur" is written in white, bold, sans-serif font in the center.

Crée une variable que l'on va appeler « Compteur ».

PUIS

Explique que devrait être le rôle d'une telle variable?

.....
.....
.....

3

Réalisation du programme sur mBlock

Utilise tout ce que tu viens de voir jusqu'à présent pour développer un programme avec une variable « compteur » qui commencera ton compte à rebours à 10 et finira à 0.

CALCULE

Si la variable « Compteur » = 10, à combien la fonction « répéter x fois » est égale ?

Même question si « Compteur » = 15, 20 ou 1000

Que vous pouvez vous en conclure ?

.....
.....
.....
.....

ACTIVITÉ : Apprendre à programmer un compte à rebours

- Corrigé

Partie 1 - Comprendre

2

Restitution

Que permet de faire un compte à rebours et pourquoi est-ce un outil très utile ?

Un compte à rebours permet de définir un intervalle de temps précis entre un commencement et une fin. C'est très utile par exemple pour déterminer le temps restant d'une action qui est en cours.

Dans quelles situations utilise-t-on un compte à rebours ? Trouve au moins 3 situations différentes.

On s'en sert presque quotidiennement : Lancement de fusées, Nouvel An, jeux télé, etc...

Selon toi, quels sont les paramètres importants pour qu'un compte à rebours fonctionne ?

Un point de départ et une fin. Dans la très grande majorité des cas la fin est égale à 0. En revanche le point de départ varie énormément.

Partie 2 – 1^{ère} méthode, manuellement

1

Parvenir à afficher un nombre sur l'écran de l'IZlone

Dans cette partie nous allons construire un compte à rebours où c'est vous qui allez faire tout le travail. Pour faire simple vous allez forcer votre IZlone à afficher successivement le chiffre 10 puis 9, 8..., jusqu'à 0.



Vous allez vite comprendre avec le corrigé.

Voilà ce que vous est demandé pour afficher un 10 sur l'écran de votre IZlone.

D'abord on demande de « Générer le code ». Enfin dans la librairie IZlone, faites glisser/déposer la fonction « Afficher le nombre x ». Renseignez la valeur 10, la couleur importe peu.

Voilà, vous avez fini. Vous pouvez déjà sentir ce que nous allons faire juste après.

2

Parvenir à faire afficher une suite de nombre à l'écran.

Très simple, continuez le programme précédent en ajoutant ou dupliquant la fonction « Afficher le nombre x »

Attention, il y a quand même un petit changement. Regardez ci-dessous.

On a dû rajouter la fonction « attendre 1 seconde ». IZlone exécute ses programmes à grande vitesse. Ici, sans cette fonction d'attente, la matrice vous afficherait bien un compte à rebours de 10 à 0, mais vous ne verriez rien. Uniquement le 0 final qui reste affiché à l'écran. Tout est allé très vite. Forcez la matrice à attendre 1 seconde avant d'exécuter l'instruction suivante.



De plus cela étalonne vous compte à rebours sur l'unité de temps qu'est la seconde.

Rien ne vous empêche de modifier le temps d'attente entre chaque ligne pour aller plus lentement ou plus rapidement.

Notez que nous avons inclus le programme dans une boucle « Répéter indéfiniment ». Cette fonction relance le compte à rebours à 10 dès qu'il atteint 0. Vous avez un compte à rebours sans fin.

Vous pouvez également voir que ce programme est à la fois long et répétitif.

Nous allons remédier à ça dans la partie suivante.

Partie 3 – 2^{ème} méthode, automatiquement

Passons à la partie la plus intéressante, réussir à automatiser son compte à rebours pour ne plus le faire à la main, car c'est assez ennuyant et peu utile en programmation.

Utiliser les variables dans ce cas vous permettra d'optimiser votre programme et votre temps.

1

Faire compter la machine

Rappelle ou explique ce qu'est une variable en programmation.

En informatique, une variable permet d'associer un nom à une valeur. Dans la plupart des langages de programmation, les variables peuvent changer de valeur au cours du temps et être dynamiques. Ce sera le cas ici pour le compte à rebours.

Nous allons associer un nom, « Compteur » à une valeur, « 10 ». Puis cette valeur changera pour devenir 9 puis 8, etc... jusqu'à 0. Si vous n'avez pas encore compris, on vous montre tout dans la dernière partie de ce corrigé.

Rappelle ou explique le rôle de la fonction « répéter x fois ».

Cette fonction permet comme son nom l'indique de répéter les instructions qui lui sont données un nombre de fois bien déterminé afin d'éviter d'écrire x fois les mêmes lignes. C'est un gain de temps et d'efficacité certains.

2

Créer une variable

Pour créer la variable « Compteur » dans mBlock, allez dans l'onglet Script > Blocs & variables > cliquez sur « Créer une variable » > renseignez « Compteur » > faites OK



Vous venez de créer votre variable ainsi que de quoi la définir et la faire varier

Explique que devrait être le rôle d'une telle variable?

Le rôle de cette variable va être de faire afficher à votre IZlone une variable dans laquelle seront contenues des valeurs qui varient.

3

Réalisation du programme sur mBlock

Voyez le résultat par vous-même. Rien de plus, rien de moins. Avouez que c'est plus rapide et plus efficace comme procédé.



Concrètement, vous définissez la valeur de votre variable. Elle prend la valeur 10 (mettre Compteur à 10). Puis vous répétez 11 fois les instructions suivantes : afficher le nombre Compteur.

Souvenez-vous du début de cette activité, vous avez déjà utilisé la fonction « Afficher le nombre x ». Cette fois « x » est égal à la variable « Compteur », soit égal à 10. Votre matrice vous affiche 10. Vous demandez à la machine d'attendre 1 seconde. Nous avons déjà vu ça. Vous soustrayez 1 à 10. Ainsi la variable « Compteur » = 10 - 1 = 9. Ainsi de suite jusqu'à 0.

Notez également la présence de la boucle « répéter indéfiniment ». Nous en avons déjà parlé plus tôt dans cette activité. Elle permet de reprendre le compteur à 10 pour avoir un compte à rebours sans fin. Enlevez cette fonction et le compteur s'arrête à 0.

Si la variable « Compteur » = 10, à combien la fonction « répéter x fois » est égale ?

Même question si « Compteur » = 15, 20 ou 1000

Que vous pouvez vous en conclure ?

Si « Compteur » = 10, « répéter x fois » sera à 11

Si « Compteur » = 15, « répéter x fois » sera à 16

Si « Compteur » = 20, « répéter x fois » sera à 21

Si « Compteur » = 1000, « répéter x fois » sera à 1001

On peut en conclure que « répéter x fois » = « Compteur » + 1

Cela pourrait vous être utile pour davantage automatiser votre compte à rebours. L'idée serait qu'importe la valeur de la variable « Compteur », votre compte à rebours arrivera toujours à 0. Et ce sans toucher à la fonction « répéter x fois ».

ACTIVITÉ : Apprendre à programmer un jeu SNAKE - Séance 1

Extensions utilisées : IZlone + Snake

Partie 1 - Comprendre

1

Le Snake, c'est quoi ?

Avant-propos : Durant cette activité répartie sur 3 séances, vous allez apprendre à programmer un Snake sur votre IZlone et sur mBlock. La difficulté sera croissante de séance en séance. Cette première séance sera très « simple » afin que tu puisses déjà t'amuser à programmer et à jouer !

Le Snake (=serpent en anglais) est un jeu où le but est de faire grandir son serpent en attrapant de la nourriture. Le serpent n'a pas le droit de se toucher, sinon le joueur perd. Il n'a pas le droit de toucher les murs, sinon le joueur perd également. Le serpent se dirige grâce aux actions du joueur.

À TOI !

À partir du descriptif du jeu, détermine les directions possibles que peut prendre le serpent

.....
.....

CONTINUE

Détermine le nombre d'actions nécessaires pour faire bouger le serpent dans tous les sens. Que devras-tu utiliser sur ton IZlone ? Décris leurs actions.

.....
.....
.....

Impression écran Snake qui va manger

Regardez ce serpent dans son environnement naturel qui s'apprête à manger sa proie afin qu'il puisse grandir !

Détermine le type de boucle dans mBlock pour programmer cette action. Comment ça marche ?

.....
.....
.....

Partie 2 – Si ... alors...

1

Liens entre l'homme et la machine

KEEP GOING

Rédige sous la forme Si... Alors... tout ce qu'il faut pour faire bouger ton serpent dans toutes les directions. (Ex : Si j'appuie sur le bouton gauche, Alors...)

.....
.....
.....

GO ON

Rédige une condition supplémentaire qui concerne la taille du serpent.

.....
.....

2

Une difficulté supplémentaire

À TOI !

Il faudra rajouter dans ton code une fonction précise qui s'appelle « renouveler la nourriture ». Dans quel bloc conditionnel vas-tu la mettre et dans quel ordre ?

.....
.....

Est-ce que l'ordre des blocs à l'intérieur d'une boucle est important ? Dans notre cas ?

.....
.....

Partie 3 – Réalisation

1

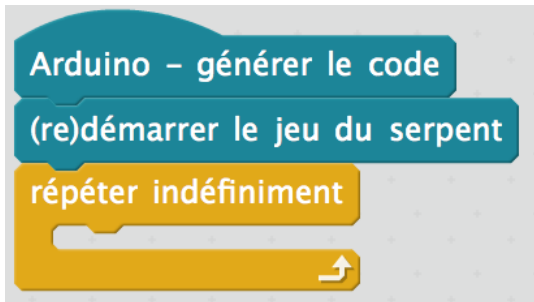
C'est l'heure de programmer sur mBlock

VA SUR MBLOCK

Lance mBlock et installe l'extension Snake et IZlone si ce n'est pas fait (retourne sur le Manuel d'Utilisateur si tu as un doute)

LANCE-TOI

Tu as tous les éléments pour réussir, alors réalise le programme qui te permettra de jouer.



Coup de pouce : ton programme devra commencer comme ceci :

Intègre le reste de ton programme dans la boucle « répéter indéfiniment ».

Cette dernière te permet de faire bouger ton serpent plus d'une fois.

2

Limites

N'OUBLIE PAS

Pour l'instant tu as programmé un Snake très simple. Il manque plusieurs conditions. C'est entre autres la raison pour laquelle tu ne pourras pas perdre (pour l'instant) avec votre Snake. Tu n'as pas encore programmé les conditions d'échecs ou de victoire.

Nous verrons tout ceci lors de la séance 2.
Sauvegarde ton travail sur ta session !

ACTIVITÉ : Apprendre à programmer un jeu SNAKE - Séance 1 - Corrigé

Partie 1 - Comprendre

1

Le Snake, c'est quoi ?

À partir du descriptif du jeu, détermine **les directions** possibles que peut prendre le serpent.

Vers le haut, bas, gauche, droite.

Il faut bien poser les bases de la réflexion autour de cet exercice de serpent et surtout bien comprendre en quoi consiste le jeu. Faire déplacer le serpent pour qu'il attrape de la nourriture afin qu'il puisse grandir au maximum.

Notez que dans le jeu du Snake, le serpent n'est pas censé revenir sur lui-même. Il s'agit ici d'un problème de physique. En revanche le programme ne se préoccupe pas de la physique rien n'empêche le serpent de revenir sur lui-même. Si ce n'est le programme lui-même. Pour nous vous facilitez la tâche, nous avons fait en sorte que le serpent ne puisse pas revenir en arrière. Donc si le serpent va vers la droite, il ne pourra pas aller vers la gauche SAUF s'il effectue un virage au préalable.

Détermine **le nombre** d'actions nécessaires pour faire bouger le serpent dans tous les sens. Que devras-tu utiliser sur ton IZlone ? **Décris** leurs actions.

Quatre. Lorsque j'appuie sur le bouton du haut, le serpent se dirige vers le haut. Lorsque j'appuie sur le bouton du bas, le serpent se dirige vers le bas. Etc...

Vous allez devoir utiliser les actionneurs de votre IZlone, c'est-à-dire les boutons. Ils sont au nombre de 5, mais vous en utiliserez que 4 aujourd'hui.

Leurs actions sont décrites comme telles : si j'appuie sur un des actionneurs, j'envoie un message à mon IZlone. Ici faire changer le serpent de direction.

Détermine **le type de boucle** dans mBlock pour programmer cette action

Boucle conditionnelle. Si... Alors... SI le serpent mange nourriture ALORS il grandit.

De la même manière vous pourrez comprendre qu'il s'agit du même type de boucle pour faire déplacer le serpent : SI bouton du haut pressé ALORS le serpent va vers le haut.

Si les conditions sont vraies, alors les instructions sous le « Si » seront exécutées.

Remarque : À ce stade, vous êtes déjà capable d'écrire et de faire un programme pour avoir un Snake très simplifié. Le serpent bouge et s'agrandit.

À ce niveau, le serpent peut « traverser » les murs et se toucher. Ces contraintes physiques seront vues plus tard lors de la séance 2 pour affiner le programme.

Partie 2 – Si ... alors...

L'intérêt de cette partie est de faire travailler l'élève sur la boucle conditionnelle Si... Alors...

L'élève pourra déduire à l'aune de la première partie (Comprendre) de l'activité l'effet de causalité. Si j'appuie sur un bouton, alors le serpent bouge.

1

Liens entre l'homme et la machine

Rédige sous la forme **Si... Alors... tout ce qu'il faut pour faire bouger ton serpent dans toutes les directions. (Ex : Si j'appuie sur le bouton gauche, Alors...)**

Si j'appuie sur le bouton de gauche, Alors le serpent tourne vers la gauche.

Si j'appuie sur le bouton de droite, Alors le serpent tourne vers la droite.

Si j'appuie sur le bouton du haut, Alors le serpent tourne vers le haut.

Si j'appuie sur le bouton du bas, Alors le serpent tourne vers le bas.

On voit bien ici l'intérêt des actionneurs et les interactions en l'homme et votre IZlone. Elle exécute ce que vous lui dites.

Rédige une condition supplémentaire qui concerne la taille du serpent.

Si le serpent mange la nourriture, alors il grandit.

Dans ce cas, la cause de l'exécution des instructions n'est plus l'homme, mais une fonction qui est dans votre programme. Lorsque cette fonction est « vraie » alors le programme exécute les instructions de la boucle conditionnelle.

2

Une difficulté supplémentaire

Il faudra rajouter dans ton code une fonction précise qui s'appelle « renouveler la nourriture ». Dans quel bloc conditionnel vas-tu la mettre et dans quel ordre ?

Il faut rajouter la fonction « renouveler nourriture » dans le bloc conditionnel lié à la taille du serpent.

Si le serpent mange la nourriture, alors :

- le serpent grandit
- renouveler la nourriture

Cette fonction permet de changer les coordonnées de la nourriture. Sinon elle réapparaît au même endroit. C'est un peu dommage.

Est-ce que l'ordre des blocs à l'intérieur d'une boucle est important ? Dans notre cas ?

En général l'ordre dans lequel les instructions sont écrites est très important, nous le verrons dans la séance 2.

Dans notre cas précis, faire renouveler la nourriture avant de faire grandir le serpent ne serait pas déroutant. Ça contrevient simplement à la bonne compréhension et à l'ordre des choses. Mais dans les faits, rien ne va changer dans l'utilisation de votre programme.

Partie 3 – Réalisation

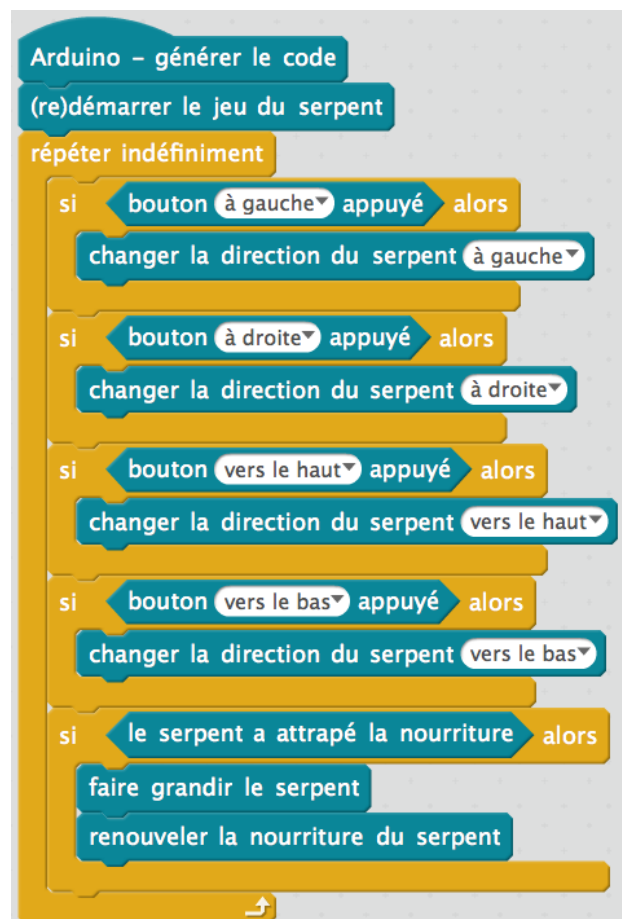
1

C'est l'heure de programmer sur mBlock

Comme vous pouvez le voir, ce programme pour le Snake est très simple. Mais il permet d'avoir une première expérience avec ce jeu et en programmation. Du moins pendant cette activité.

Dans ce programme nous avons :

- « Arduino - générer le code ». Il faut toujours avoir ce bloc en tête de votre programme principal pour ensuite téléverser le code dans votre IZlone. **Si vous ne savez pas comment faire, consulter notre manuel d'utilisation complet.**
- la fonction « (re) démarrer le jeu du serpent » est assez explicite. Elle gère le démarrage du jeu. Nous travaillerons dessus un peu plus lors de la séance 2.
- la boucle « répéter indéfiniment » est là pour permettre à votre programme de pouvoir exécuter plusieurs fois les différentes boucles conditionnelles à l'intérieur. Et vous aurez besoin de faire bouger votre serpent vers le haut plusieurs fois par exemple.



Maintenant c'est à vous de jouer, littéralement.

En revanche après quelques essais vous allez peut-être ressentir de la frustration. En effet avec ce programme, vous ne pouvez pas perdre et le jeu est très, voire trop facile.
Ne vous inquiétez pas, tout va se corser dans la séance 2 où l'on va complexifier ce programme.

ACTIVITÉ : Apprendre à programmer un jeu SNAKE - Séance 2

Extensions utilisées : IZlone + Snake

Durant cette séance 2, nous allons rajouter des composantes à notre programme initial réalisé lors de la séance 1.

Pour commencer, récupérer le programme mBlock que tu as déjà fait. Nous allons partir de cette base. Nous partons du principe que tu as un programme identique à celui fourni dans le corrigé de la séance 1. Si vous n'êtes pas sûr de vous, nous vous invitons à récupérer le corrigé.

Sinon c'est parti !

Nous allons complexifier le programme du Snake étape par étape.

À la fin, tu auras un programme intéressant.

Partie 1 - Initialisation

1

Variables

POURSUIS

Rappelle ce qu'est une variable en programmation.

RETOURNE SUR MBLOCK

Crée 2 variables dans mBlock. Une appelée « TailleMax » et un autre « Score ».

Sous « Arduino - générer le code » mets la première variable à 10 et la seconde à 0

Explique pourquoi on associe ces valeurs à nos variables.

2

Démarrage du programme

CONTINUE

Fais en sorte qu'au démarrage ton serpent commence en allant vers le bas (du haut vers le bas).

DON'T STOP

Fais en sorte que ton programme attende jusqu'à ce que « commencer » soit relâché pour démarrer le Snake.

N'hésite pas à tester ton programme pour vérifier que l'initialisation est correcte.

Partie 2 – Victoire ou défaite

1

Les contraintes physiques

KEEP GOING

Liste l'ensemble des contraintes physiques qui s'exercent sur le serpent (Coup de pouce : il y en a 3. Dont une que tu connais déjà, il ne peut pas revenir sur lui-même).

.....

.....

.....

GO ON

Explique ce qu'il se passe si le serpent est confronté à ces limites.

.....

.....

.....

À TOI

Combien y a-t-il de cas où le joueur perd ? Est-ce que le joueur peut gagner ?

.....

.....

```

si [le serpent a foncé dans le mur] ou [le serpent s'est foncé dedans] alors
  [Score] le joueur a perdu
  [afficher le nombre]
  [mettre Score à 0]
  [bouton commencer relâché]
  [attendre jusqu'à]
  [(re)démarrer le jeu du serpent]
  
```

Ci-contre, tu as tous les éléments pour gérer les cas où le joueur perd. Retourne sur mBlock et ordonne les instructions pour que le programme fonctionne correctement.

Le jeu Snake a pour réputation d'être un jeu très difficile à finir. Alors on a décidé de créer un programme pour le jeu où l'on peut gagner (quasiment) à tous les coups.

Voici une partie du programme du Snake. C'est elle qui permet au joueur de gagner.

Refais le programme et complète-le (ligne 2,5, 7) et explique ce qu'il se passe ligne par ligne.

```

si [le serpent a attrapé la nourriture] alors
  [ajouter à Score]
  [faire grandir le serpent]
  [renouveler la nourriture du serpent]
  si [taille du serpent = ] alors
    [le joueur a gagné]
    [afficher le nombre]
    [attendre jusqu'à bouton commencer relâché]
    [mettre Score à 0]
    [(re)démarrer le jeu du serpent]
  
```

.....

.....

.....

.....

.....

.....

.....

.....

.....

1

Assemblage de tous les blocs

SUR MBLOCK

Assemble tous les morceaux de programme réalisés jusque-là pour faire fonctionner ton code. Tu devrais avoir un serpent d'une taille de 10 pixels maximum avec un score de 7.

Tu peux augmenter la difficulté en augmentant la valeur de la variable « TailleMax ».

Tu peux encore accélérer ton serpent dès qu'il mange la nourriture pour pimenter davantage ton jeu.

Autrement c'est la fin de la séance 2.

ACTIVITÉ : Apprendre à programmer un jeu SNAKE - Séance 2 - Corrigé

Partie 1 - Initialisation

1

Variables

Rappelle ce qu'est une variable en programmation.

Une variable en programmation permet d'associer une valeur à un nom. On va pouvoir y stocker une valeur.

Crée 2 variables dans mBlock. Une appelée « TailleMax » et un autre « Score ».



Allez dans l'onglet Script > Blocs & variables > Créer une variable > renseignez les noms > faites OK.

Notez bien que vous allez créer en même temps les fonctions « mettre x à y », « ajouter à x y », « montrer la variable x » et « cacher la variable x ».

Ces fonctions apparaissent à la création d'un variable. Elles vous seront utiles.

Sous « Arduino - générer le code » mets la première variable à 10 et la seconde à 0



Rien de compliqué ici. Faites glisser les fonctions « mettre x à y » et choisissez la variable adéquate.

Explique pourquoi on associe ces valeurs à nos variables.

La variable « Score » va vous servir de compteur de point. Dès que vous allez manger un morceau de nourriture, on fera +1 au score. Comme vous commencez avec 0 point et qu'il faut les accumuler, on stocke la valeur 0 dans « Score ».

Pour la variable « TailleMax », nous la mettons à 10 arbitrairement. Cette variable stocke le nombre 10, et dès que le serpent aura une taille de 10 pixels, nous ferons gagner le joueur. Mais tout ça reste à programmer. Il faudra rajouter quelques instructions plus tard.

2

Démarrage du programme

Fais en sorte qu'au démarrage ton serpent commence en allant vers le bas (du haut vers le bas).



Très facile aussi.

Fais en sorte que ton programme attende jusqu'à ce que « commencer » soit relâché pour démarrer le Snake.



Un peu plus complexe. Il faut que vous utilisiez la fonction « attendre jusqu'à » et y insérer la fonction IZlone « bouton commencer relâché ». Dès lors votre IZlone attendra d'exécuter le reste des instructions tant que vous n'avez pas appuyé sur le bouton Start. Et c'est exactement l'effet que l'on recherche.

Partie 2 – Victoire ou défaite

1

Les contraintes physiques

Liste l'ensemble des contraintes physiques qui s'exercent sur le serpent. (Coup de pouce : il y en a 3)

Le serpent ne peut pas toucher les murs, le serpent ne pas se toucher lui-même, le serpent ne peut pas revenir sur lui-même.

Lors de la séance 1, nous ne nous étions pas préoccupés de ces contraintes physiques. C'est pour ça que votre serpent pouvait se toucher et traverser les murs. Maintenant on essaie de rectifier ceci pour que notre Snake ressemble un peu plus au « vrai » Snake.

Explique ce qu'il se passe si le serpent est confronté à ces limites.

Il faut que vous imaginiez quel serait l'ensemble des possibilités si les contraintes physiques sont atteintes. Et bien le joueur perd. On introduit la condition d'échec. Notez que le joueur perd uniquement lorsque le serpent se touche ou touche les murs. Le serpent ne peut pas revenir sur lui-même et c'est tout. Pas de défaite dans ce cas.

Cependant

À ce moment de l'activité, si vous veniez à coder le programme, les contraintes physiques ne seraient pas si contraignantes. En effet si le serpent touche un mur, il réapparaît de l'autre côté. Si le serpent se touche lui-même, il se traverse. En effet votre code n'a pas suffisamment changé depuis la fin de la séance 1. Vous en êtes presque au même point. Il vous sera possible de modifier l'environnement du programme en laissant ou non certaines contraintes. Nous allons voir ça très rapidement dans cette séance.

Évidemment, ici nous allons définir une défaite.

Combien y a-t-il de cas où le joueur perd ? Est-ce que le joueur peut gagner ?

Le joueur perd dans 2 cas. (et non 3)

-si le serpent touche les murs

-Si le serpent se touche lui même

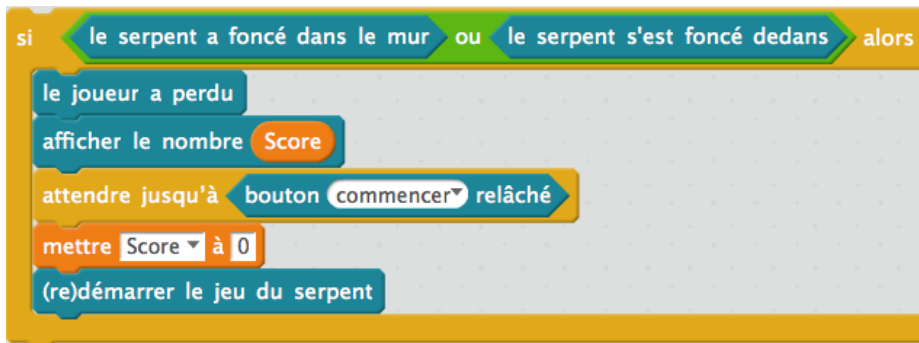
(- le fait que le serpent ne puisse pas revenir sur lui-même n'est pas une condition de défaite, simplement une condition limitative de mouvement, le serpent ne revient pas en arrière et c'est tout).

Dans le cas d'une victoire :

Souvenez-vous que nous avons créé une variable « TailleMax » et qu'elle est à 10. On peut imaginer que si la taille du serpent = 10 alors le joueur gagne.

Vous n'avez pas encore programmé de conditions de victoire, même si vous parveniez à remplir toutes les cases de la matrice à LED avec votre serpent, vous serez confronté à une contrainte physique : le serpent ne peut pas se toucher. Et cette contrainte, nous la mettons en place tout de suite.

Retourne sur mBlock et ordonne les instructions pour que le programme fonctionne correctement.



Voilà, vous pouvez enfin perdre !!!

Remarquez la fonction verte « ou » ici présente, elle vous permet d'éviter de rédiger 2 fois la fonction Si...

Alors.... Vous regroupez en une seule les deux contraintes qui impliquent une défaite du joueur. Donc si l'une des conditions est vraie, votre IZlone exécutera les instructions suivantes dans l'ordre :

-la fonction « le joueur a perdu » s'exécute, elle arrête le programme.

-Vous demandez à afficher le nombre stocké dans la variable « Score ». Votre score apparaît sur la matrice.

Note : Nous verrons dans la partie du programme concernant la victoire comment programmer l'augmentation de votre score qui est censé être à 0 en début de programme.

-Votre IZlone attend que vous appuyiez sur le bouton Start. Nous avons déjà vu ces instructions à l'initialisation.

-Votre IZlone remet votre score à 0. Vous n'allez pas repartir du score précédent qui est toujours stocké.

-Le jeu redémarre.

Avec ces instructions vous avez enfin un Snake digne de ce nom où vous pouvez perdre. Sympa.

Voici une partie du programme du Snake. C'est elle qui permet au joueur de gagner. Refais le programme et complète-le (ligne 2,5, 7) et explique ce qu'il se passe ligne par ligne.



Allons-y pas à pas, vous connaissez le début :

- rien de nouveau dans les 4 premières lignes hormis « ajouter à Score 1 ». C'est assez clair, dès que vous attrapez de la nourriture, votre programme rajoute 1 à la variable « Score ». C'est ainsi que vous augmentez votre score et c'est pourquoi vous deviez le remettre à 0 dans l'exercice précédent.

- On imbrique une nouvelle condition Si... Alors.

Dans celle-ci on définit clairement quand le joueur gagne. La fonction « taille du serpent » joue le rôle d'une variable. Elle stocke toute seule la taille en pixel du serpent. Quand ce dernier atteint la « TailleMax », le programme exécute le reste des instructions. Souvenez-vous que nous avons défini « TailleMax » à 10, mais vous pouvez tout à fait changer.

- Le reste du programme est assez clair, il est similaire à ce que nous avons vu dans l'exercice précédent. Notez simplement que l'on fait appel à la fonction « le joueur a gagné » dans ce cas-là.

Partie 3 - Réalisation

1

Assemblage de tous les blocs

Vous avez fait le plus dur. Le reste c'est du plaisir.

À présent vous avez un Snake qui correspond en tout point à un « vrai ».

Voir le code final page suivante.

```

Arduino - générer le code
mettre TailleMax à 10
mettre Score à 0
changer la direction du serpent vers le bas
attendre jusqu'à bouton commencer relâché
(re)démarrer le jeu du serpent
répéter indéfiniment
  si bouton à gauche relâché alors
    changer la direction du serpent à gauche
  si bouton à droite relâché alors
    changer la direction du serpent à droite
  si bouton vers le haut relâché alors
    changer la direction du serpent vers le haut
  si bouton vers le bas relâché alors
    changer la direction du serpent vers le bas
  si le serpent a foncé dans le mur ou le serpent s'est foncé dedans alors
    le joueur a perdu
    afficher le nombre Score
    attendre jusqu'à bouton commencer relâché
    mettre Score à 0
    (re)démarrer le jeu du serpent
  si le serpent a attrapé la nourriture alors
    ajouter à Score 1
    faire grandir le serpent
    renouveler la nourriture du serpent
    si taille du serpent = TailleMax alors
      le joueur a gagné
      afficher le nombre Score
      attendre jusqu'à bouton commencer relâché
      mettre Score à 0
      (re)démarrer le jeu du serpent

```

ACTIVITÉ : Apprendre à programmer un dessin - Séance 1

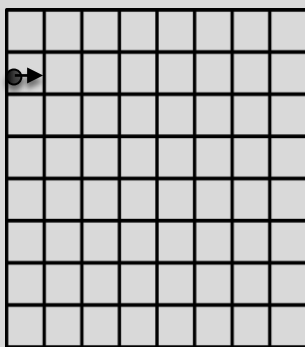
Cette première séance ne nécessite pas l'utilisation de votre IZlone et de mBlock.
Vous allez simplement commencer cette fiche et un crayon.

Attention : Le point de départ correspond au point ●
Le sens de déplacement se fait dans le sens de la flèche →

Partie 1 - Dessiner au crayon le parcours correspondant à la description de chaque programme

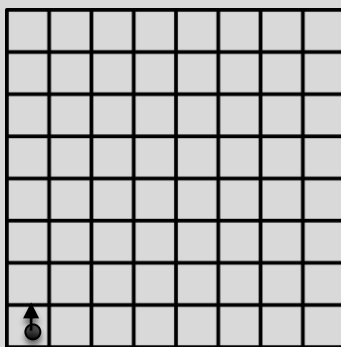
Programme de dessin 1

Avancer de 5 carreaux.
Tourner à droite de 90°.
Avancer de 2 carreaux.
Tourner à droite de 90°.
Avancer de 5 carreaux.
Tourner à droite de 90°.
Avancer de 2 carreaux.
Tourner à droite de 90°.



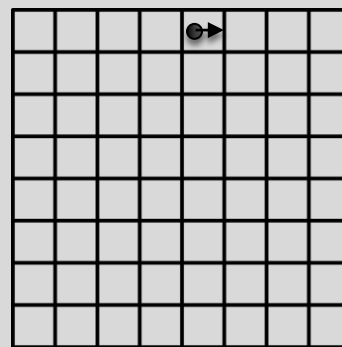
Programme de dessin 2

Avancer de 5 carreaux.
Tourner à droite de 90°.
Avancer de 4 carreaux.
Tourner à droite de 90°.
Avancer de 3 carreaux.
Tourner à droite de 90°.
Avancer de 2 carreaux.
Tourner à droite de 90°.
Avancer de 1 carreau.



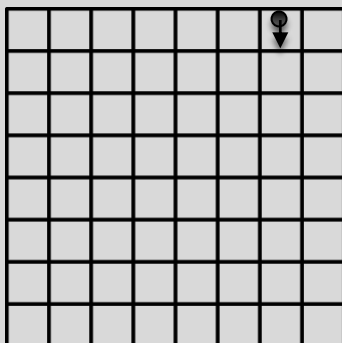
Programme de dessin 3

Reculer de 3 carreaux.
Tourner à droite de 90°.
Avancer de 3 carreaux.
Tourner à gauche de 90°.
Avancer de 3 carreaux.
Tourner à droite de 90°.
Avancer de 3 carreaux.
Tourner à droite de 90°.
Avancer de 3 carreaux.
Tourner à gauche de 90°.
Reculer de 3 carreaux.



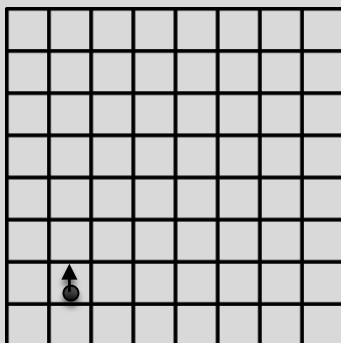
Programme de dessin 4

avancer de 1
tourner ↺ de 90 degrés
avancer de 2
tourner ↻ de 90 degrés
avancer de 3



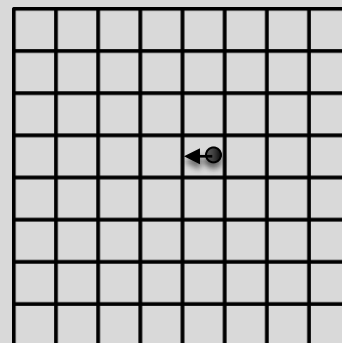
Programme de dessin 5

répéter 4 fois
avancer de 2
tourner ↺ de 90 degrés



Programme de dessin 6

répéter 4 fois
avancer de 3
avancer de -3
tourner ↺ de 90 degrés



Partie 2 - Écrire dans les blocs vides les instructions manquantes pour tracer le dessin 7

Programme de dessin 7

avancer de

tourner ↶ de degrés

Partie 3 - Écrire un programme permettant de tracer votre âge

À vous de choisir la forme précise de vos chiffres, ils doivent cependant avoir une longueur maximale de 8 carreaux et une largeur maximale de 4.

Pour vous aider, tracer les lettres dans les grilles suivantes et noter les programmes correspondant en dessous après avoir choisi un point de départ de tracé.

1 ^{er} chiffre	2 ^{ème} chiffre

ACTIVITÉ : Apprendre à programmer un dessin - Séance 2

Extensions utilisées : IZlone + Pencil

Premiers programmes avec mBlock

Objectifs : construire avec le logiciel mBlock les dessins de la séance 2 avec votre IZI one

Attention : n'oublie pas de vérifier à chaque fois sur mBlock si la carte sélectionnée est bien **Arduino Uno** ».

IMPORTANT : L'origine (1 ; 1) de la matrice est : $x = 5$, $y = 5$ en partant d'en bas à gauche.

1. Lancer le programme mBlock. Ouvrir le « Dessin n°1-IZlone.sb2 ».

Ajouter des blocs à ceux déjà existants (**qu'il est interdit de modifier**) pour que la matrice trace le dessin n°1 lorsque l'on génère le programme sur la IZlone.

Une fois que tu es satisfait de ton programme, sauvegarde ton travail dans ta session.

2. Refaire de même avec le « Dessin n°2-IZlone ».

La partie fournie par ton professeur n'est pas suffisante, il faut la compléter. Quelle instruction initiale est manquante ?

.....

3. Refaire de même avec le « Dessin n°3-IZlone ».

La partie fournie par ton professeur n'est pas suffisante, il faut la compléter. Quelle instruction initiale est-elle manquante cette fois

.....
.....

4. Refaire de même avec le « Dessin n°4-IZlone ».

La partie fournie par ton professeur n'est pas suffisante, il faut la compléter. Quelle instruction initiale est manquante cette fois ?

.....
.....

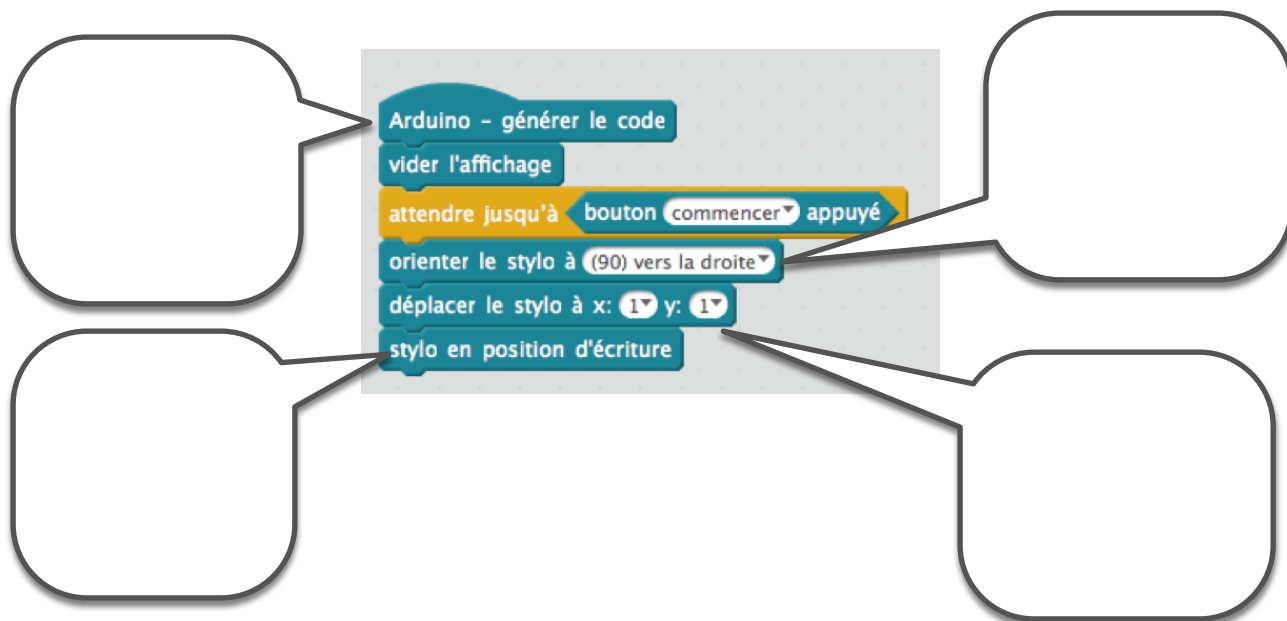
5. Pour le dessin n°5, les conditions initiales sont fournies.

Modifie-les pour que la construction se fasse dans le bon sens.

6. Pour le dessin n°6, les conditions initiales sont fournies.

Modifie-les pour que la construction se fasse et puisse être visible en entier.

7. Compléter le schéma suivant en indiquant l'utilité de chaque bloc :



8. Pour le dessin n° 7, c'est à toi de créer la quasi-totalité du programme.

Pense à utiliser tous les blocs nécessaires pour que le tracé se fasse avec des LED de couleurs différentes à chaque changement de direction.

Attention : les conditions initiales sont à modifier et une brique doit être remplacée par une autre.

9. Pour les plus rapides, écrire un programme permettant de tracer votre âge.

À vous de choisir la forme précise de vos chiffres, ils doivent cependant avoir une largeur maximale de 4 carreaux. Ouvrir le programme *âge.sb2* et y construire les deux chiffres.

ACTIVITÉ : Apprendre à programmer un dessin - Séance 3

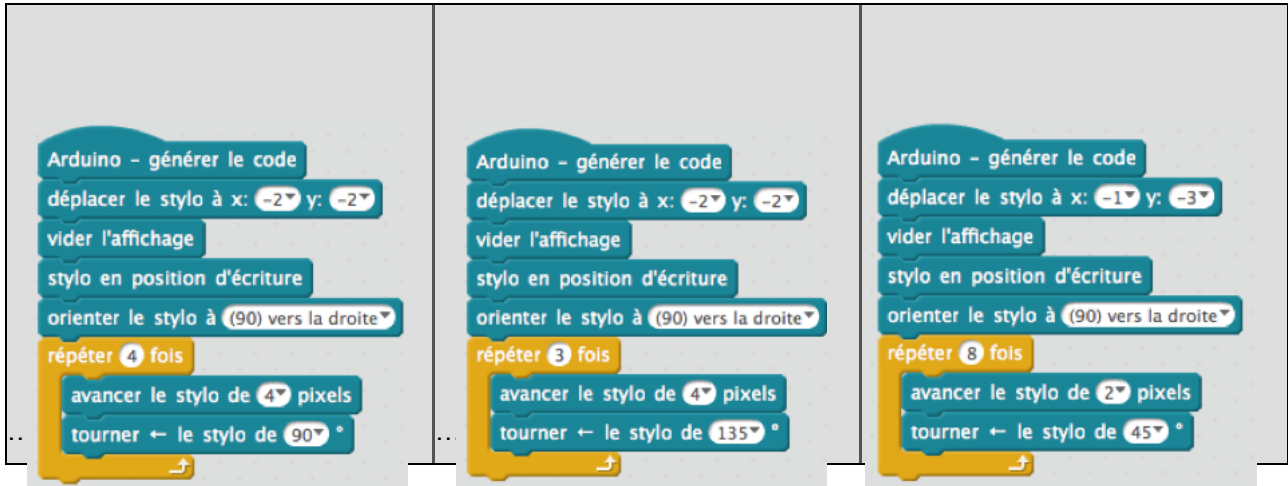
Extensions utilisées : IZlone + Pencil

Premiers programmes avec mBlock

Objectifs : continuer sur la lancée de la séance 2 avec d'autres exercices.

RAPPEL IMPORTANT : L'origine (1 ; 1) de la matrice est : x = 5, y = 5 en partant d'en bas à gauche.

1. Note sous chaque programme le nom de la figure obtenue :

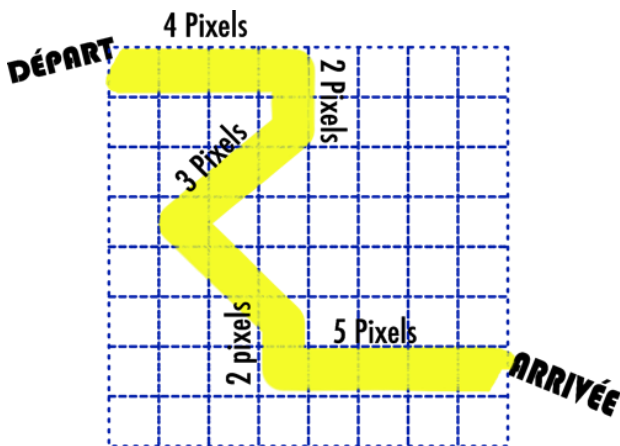


Tu n'es pas obligé de créer les programmes, une observation seule ou une simulation sur papier peut suffire.

2. Lancer le programme mBlock.

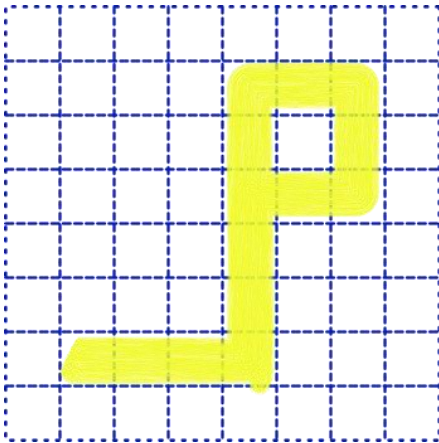
Ouvrir le « Dessin-seance3-ex2.sb2 ».

Corriger ce programme pour que l'IZlone trace la figure suivante :



Aucun bloc ne doit être ajouté ou supprimé, mais il faut seulement modifier ceux déjà présents. Une fois que tu es satisfait de ton programme, enregistre-le sur ta session.

3. À partir du fichier « Dessin-seance3-ex3.sb2 », trace le motif suivant en choisissant comme point de départ le pixel (-3 ; -3).

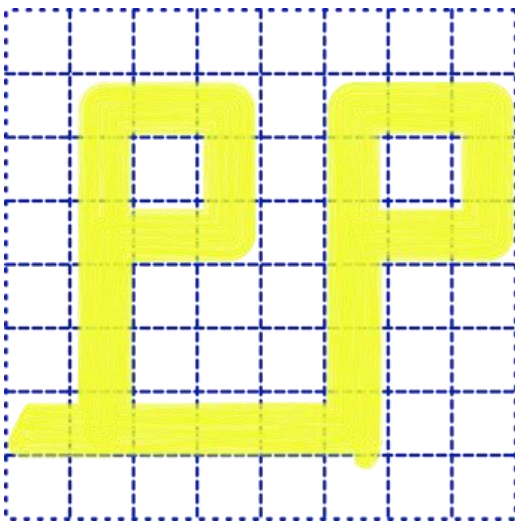


Tous les angles du motif sont droits.

Une fois que tu es satisfait de ton programme, enregistre-le sur ta session.

4. Pour les plus rapides.

Modifie ton programme « *Dessin-seance3-ex3.sb2* » pour obtenir la frise suivante : l'objectif est de réaliser cette frise avec la fonction mBlock « Contrôle » : « répéter 2 fois ». C'est-à-dire à l'initialisation de votre programme, après le bloque « stylo en position d'écriture », tout doit tenir dans la boucle « répéter 2 fois ».



Astuce : lors de l'initialisation, la coordonnée du pixel x est en dehors de la matrice.

Une fois que tu es satisfait de ton programme, enregistre-le sur ta session.

ACTIVITÉ : Apprendre à programmer un radar de recul - Séance 1

Extension utilisée : IZlone

Partie 1 - Comprendre

1

Un radar de recul avec IZlone, c'est quoi ?

Avant-propos : Pendant ces séances vous allez programmer un simulateur de radar de recul sur votre IZlone. Chaque séance permet d'apporter un morceau supplémentaire du programme. Il est important de commencer par la séance 1 et de bien suivre l'ordre des séances.

Un radar de recul dans une voiture vous permet de connaître la distance restante entre l'arrière du véhicule et le mur lors d'un stationnement par exemple.

Dans cet exercice c'est exactement ce que vous allons programmer. Nous allons simuler une voiture sur votre IZlone qui va se déplacer de gauche à droite pour se rapprocher d'un mur situé sur le bord gauche de votre matrice. La voiture est un bloc de 6 pixels, 3 de largeur et 2 de hauteur située sur la ligne 4 et 5. Le mur sera situé sur toute la colonne 1. L'objectif de ces séances est d'arriver à faire clignoter plus ou moins vite le mur en fonction de la distance de ce dernier avec la voiture.

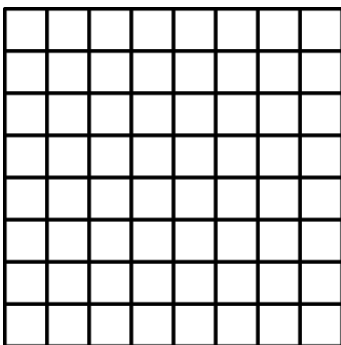
-Situation de départ : la voiture est verte et accolée au bord droit de ton IZlone et le mur clignote lentement en vert.

-Lorsqu'il y a 1 pixel d'écart entre le mur de gauche et la voiture, le mur clignote plus rapidement en orange.

-Lorsque la voiture est au contact du mur de gauche, ce dernier clignote très rapidement en rouge.

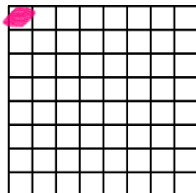
À TOI !

D'après l'énoncé reproduis sur le quadrillage ci-dessous la situation de départ.



Utilise un crayon à papier et rajoute des annotations pour la couleur et le clignotement.

Note : Dans cette activité, l'origine (1,1) de la matrice est le point suivant.



← Origine de la matrice

Partie 2 - Programmation

1

La voiture

Va sur mBlock pour commencer à programmer la voiture.

CONTINUE

Renseigne l'ensemble des coordonnées où se situe la voiture au départ, il en faut 6. Rédige-les selon la manière de l'exemple : (6,4) etc... Commence par le numéro de la colonne puis de la ligne.

.....
.....
.....

VA SUR MBLOCK

Crée un programme pour faire afficher ta voiture dans la situation de départ. Utilise ce genre de bloc pour ton programme :



Dès que tu auras vérifié le résultat sur mBlock, il te sera difficile de faire bouger ta voiture sans devoir à nouveau téléverser ton programme autant de fois que tu veux faire bouger ta voiture. C'est pourquoi nous allons avoir besoin de **2 variables**.

RÉDIGE

Explique ce qu'est une variable en programmation.

.....
.....
.....

AJOUTE

Dans mBlock, crée 2 variables, l'une appelée « ligne_voiture » et l'autre « colonne_voiture » comme ci-dessous.

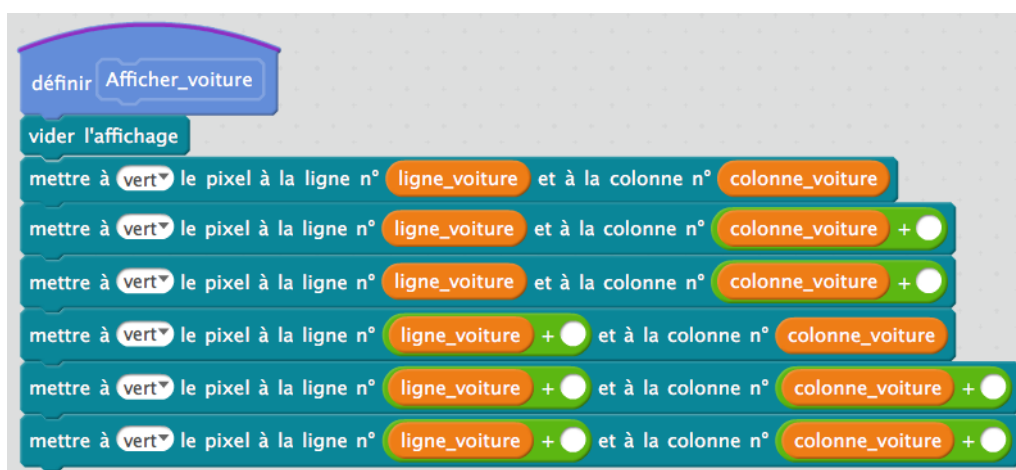


Puis crée le bloc « Afficher_voiture » comme ci-dessous. Cette fonction nous permettra de demander à ton IZlone d'afficher la voiture.



COMPLÈTE

Ci-dessous voici la fonction entière qui te permet de faire afficher ta voiture sur l'écran de ton IZlone. En revanche la fonction n'est pas complète. Il faut que tu renseignes sur cette feuille puis dans mBlock les cases vides avec les bonnes valeurs pour que ta voiture apparaisse aux coordonnées de la situation de départ.



Astuce 1 : les 3 premières lignes « mettre à vert les pixels... » servent à allumer les pixels de la voiture à la ligne 4. Les 3 dernières lignes pour les 3 pixels ligne 5.

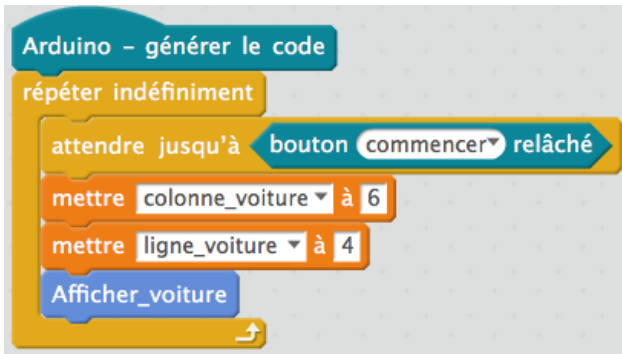
Astuce 2 : il faut compléter ce programme soit avec le chiffre 1 soit avec le chiffre 2.

Selon toi, explique pourquoi il y a la fonction « vider l'affichage » au début du programme?

.....
.....
.....

VÉRIFIE

Contrôle que ta fonction « Afficher_voiture » fonctionne correctement en rajoutant à coté de ton programme, le morceau ci-dessous. Puis téléverse tout le programme pour le tester.



2

Conserve ton programme

SAUVEGARDE

Sauvegarde sur ta session tes morceaux de programme. Ils te seront utiles pour la séance 2.

C'est la fin de la séance 1. Lors de la séance 2, nous créerons les morceaux de programme pour faire afficher les murs et nous commencerons à faire bouger la voiture grâce aux boutons de ton IZlone.

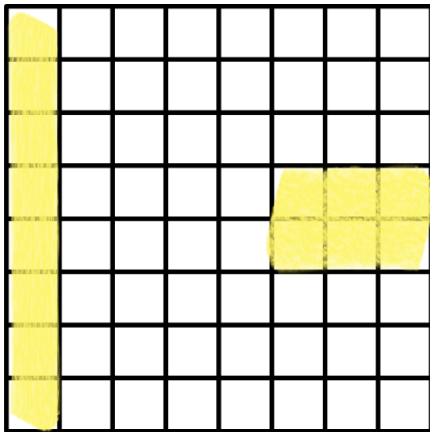
ACTIVITÉ : Apprendre à programmer un radar de recul - Séance 1 - Corrigé

Partie 1 - Comprendre

1

Un radar de recul avec IZlone, c'est quoi ?

D'après l'énoncé reproduit sur le quadrillage ci-dessous la situation de départ.



Comme écrit dans l'énoncé, le mur de gauche se situe sur toute la colonne 1. Le pixel le plus en haut à gauche est l'origine de la matrice (1,1).

La première valeur des coordonnées représente la position sur l'axe des abscisses. La seconde valeur pour les ordonnées.

< — Ligne 4

< — Ligne 5

Partie 2 - Programmation

1

La voiture

Renseigne l'ensemble des coordonnées où se situe la voiture, il en faut 6. Rédige-les selon la manière de l'exemple : (6,4) etc... Commence par le numéro de la colonne puis de la ligne.

L'ensemble des coordonnées de la voiture lors de la situation de départ est :

(6,4), (7,4), (8,4), (6,5), (7,5), (8,5)

Crée un programme pour faire afficher ta voiture dans la situation de départ. Utilise ce genre de bloc pour ton programme :



Voilà le petit programme à réaliser.

Rien de très dur, mais des choses à ne pas oublier :

– Vider l'affichage permet d'enlever le logo « Z » de la matrice qui est présent au démarrage. Si vous l'oubliez votre voiture risque de s'y superposer.

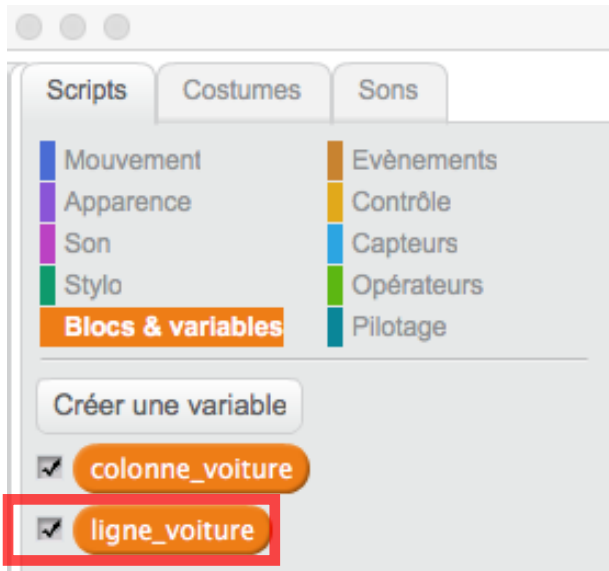
– Ne pas reprendre automatiquement les coordonnées rédigées de la question précédente. Sur mBlock on commence par renseigner l'ordonnée.

Explique ce qu'est une variable en programmation.

En informatique, une variable permet d'associer un nom à une valeur. Dans la plupart des langages de programmation, les variables peuvent changer de valeur au cours du temps et être dynamique. Ce sera le cas ici pour le radar de recul.

Nous allons associer un nom, « colonne_voiture » à une valeur, « 6 ». Puis cette valeur changera pour devenir 5 puis 4, etc... . Si vous n'avez pas encore compris, on vous détaillera tout ceci un peu plus tard.

Dans mBlock, crée 2 variables, l'une appelée « ligne_voiture » et l'autre « colonne_voiture » comme ci-dessous.



Pour créer une variable dans mBlock, allez dans l'onglet Scripts > menu Blocs & variables > cliquez sur « Créer une variable (encadré en rouge ci-dessus) > une boîte de dialogue s'ouvre > renseignez le nom de la variable > faites OK.

Puis crée le bloc "Afficher_voiture" comme ci-dessous. Cette fonction nous permettra de demander à ton IZlone d'afficher la voiture.



Pour créer un bloc dans mBlock, allez dans l'onglet Scripts > menu Blocs & variables > cliquez sur "Créer un bloc" (encadré en rouge ci-dessus) > une boîte de dialogue s'ouvre > renseignez le nom du bloc > faites OK > le bloc apparaîtra à l'endroit où vous programmez dans mBlock.

Ci-dessous voici la fonction entière qui te permet de faire afficher ta voiture sur l'écran de ton IZlone.



– Dans ce sous-programme, qui consiste à allumer la voiture vous allez tout d’abord vider l’affichage. Nous allons vous expliquer le pourquoi de ce bloc dans la question suivante.

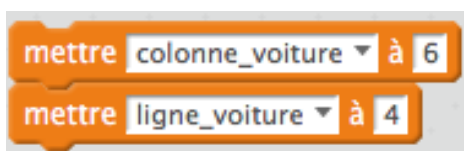
– Ensuite vous demandez à votre IZlone d’allumer un pixel en vert à la ligne “ligne_voiture” et à la colonne “colonne_voiture”. Votre IZlone va donc allumer **le pixel aux coordonnées contenues dans les 2 variables**. Si la variable “ligne_voiture” = 4 et “colonne_voiture” = 6 alors votre IZlone allumera le pixel (6,4). Pour l’instant nous n’avons pas encore attribué de la valeur à nos variables. C’est pour ça que cela peut être difficile de se représenter la chose.

– À la ligne suivante, on va simplement demander à votre IZlone d’allumer le pixel de la colonne d’à côté. C’est à ça que sert le “+1”. En effet à la ligne précédente, en exemple, nous avons dit “colonne_voiture = 6”. Et bien désormais votre IZlone va allumer la colonne 6 + 1, soit la colonne 7. Ainsi vous avez 2 pixels allumés, côte à côte aux coordonnées suivantes : (6,4) et (7,4).

Note : votre programme ne changera pas la valeur de la variable “colonne_voiture”. Elle est toujours “égale” à 6 dans notre exemple. Votre IZlone comprend très bien qu’il faut allumer à la colonne “6” + 1. Votre IZlone stocke toujours “6”.

– Dans la ligne suivante vous allez demander à votre IZlone d’allumer un pixel à la ligne “4” et à la colonne “6” + 2. Les coordonnées du nouveau pixel sont donc (8,4). Ainsi vous avez réussi à allumer les 3 pixels de la ligne 4. Vous avez la première moitié de votre voiture.

Pour donner une valeur à vos variables, vous devez utiliser les blocs suivants dans votre programme :



Nous les avons remplis avec les valeurs de notre exemple.

– Ensuite nous réalisons exactement la même opération que les 3 lignes différentes, mais cette fois nous demandons au programme d’allumer la ligne “valeur contenue dans ligne_voiture” + 1. Cette opération nous permet d’allumer la ligne 5 sur la valeur contenue dans la variable “ligne_voiture” est 4 comme dans l’exemple utilisé jusqu’à présent.

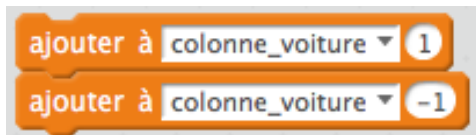
Remarque : Dans la fiche activité, il est mentionné 2 astuces pour réaliser l’exercice. Nous venons d’expliquer pourquoi il faut remplir la fiche avec des 1 et des 2. C’est pour simplement donner les bonnes coordonnées à allumer pour votre IZlone. Quant à la construction de la voiture, nous avons pris la décision de la faire ligne par ligne. Vous pouvez tout à fait changer la façon dont la voiture se construit dans mBlock. Mais il faudra alors changer le sous-programme “Afficher_voiture”.

Selon toi, explique pourquoi il y a la fonction “vider l’affichage” au début du programme?

Chose promise, chose due :

Nous utilisons ce bloc pour supprimer “l’ancienne image” de la voiture. Cette astuce permet d’avoir une voiture de toujours 6 pixels. Sans cette fonction, quand vous avancerez votre voiture vers la gauche, votre voiture sera représentée entre la colonne 5 et la ligne 8. Soit 4 pixels de largeur. En effet votre programme fait exactement ce que vous lui demandez. Il allume les pixels aux coordonnées indiquées. Cependant vous ne lui demandez pas **déteindre les pixels** où la voiture n’est plus censée être. Vous résolvez ce problème grâce à la fonction “vider l’affichage”. Votre IZlone supprime l’ancienne voiture et en recrée une aux nouvelles coordonnées.

Vous pouvez faire déplacer votre voiture en modifiant les valeurs des variables qui composent ces coordonnées (voir question précédente) grâce à ces blocs :



Si comme précédemment la variable “colonne_voiture” = 6, avec ce premier bloc, la variable passe à 7. Donc toute la voiture va d’une colonne vers la droite. Avec le second bloc (ajouter -1), la variable passe à 5. Toute la voiture va à gauche.

C’est par ce moyen que vous modifiez les valeurs de vos variables. Et pas avec les opérations “+ 1” ou “+ 2” que nous avons vu dans la question précédente.

Contrôle que ta fonction “Afficher_voiture” fonctionne correctement en rajoutant à côté de ton programme, le morceau ci-dessous. Puis téléverse tout le programme pour le tester.



Jusqu’à présent nous travaillons dans le sous-programme “Afficher_voiture”. Maintenant avec le programme ci-dessus, nous travaillons dans le programme principal. C’est dans celui-ci que nous appelons le sous-programme. C’est la dernière ligne du programme en bleu.

Au-dessus vous retrouvez les fonctions qui permettent de définir les valeurs incrémentées dans vos variables. Il s’avère que l’exemple que nous prenons jusqu’à présent pour comprendre ce qu’il se passe dans notre programme est bien ce qu’il faut faire pour réaliser cette activité.

Notre programme ne démarrera que lorsque vous aurez appuyé sur le bouton “commencer”.

Enfin tout votre programme se répétera à l’infini.

Ce programme vous permet de bien vérifier que votre voiture s’affiche correctement. Car il fallait définir les valeurs des variables pour faire exister la voiture. C’est désormais chose faite.

Cette séance est terminée, n’oubliez pas de sauvegarder votre travail.

ACTIVITÉ : Apprendre à programmer un radar de recul - Séance 2

Extension utilisée : IZlone

Partie 1 - Comprendre

1

Un radar de recul avec IZlone, c'est quoi ?

Avant-propos : Pendant ces séances vous allez programmer un simulateur de radar de recul sur votre IZlone. Chaque séance permet d'apporter un morceau supplémentaire du programme. Il est important de commencer par la séance 1 et de bien suivre l'ordre des séances.

Lors de cette séance 2, nous allons construire les morceaux de programme pour afficher le mur de gauche dans différentes couleurs et qui clignote à des rythmes différents.

Rappel des situations:

- Situation de départ : la voiture est verte et accolée au bord droit de ton IZlone et le mur clignote lentement en vert.
- Lorsqu'il y a 1 pixel d'écart entre le mur de gauche et la voiture, le mur clignote plus rapidement en orange.
- Lorsque la voiture est au contact du mur de gauche, ce dernier clignote très rapidement en rouge.

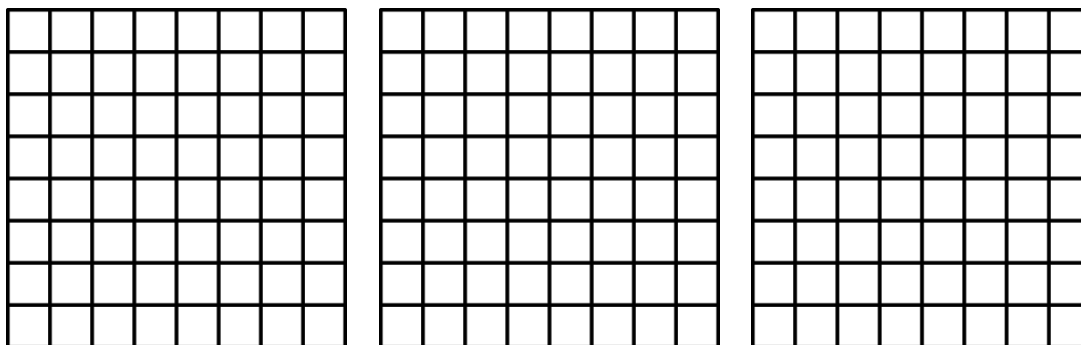
Puis nous allons créer un programme temporaire pour permettre à la voiture de se déplacer d'avant en arrière.

Récupère ton travail réalisé lors de la séance 1.

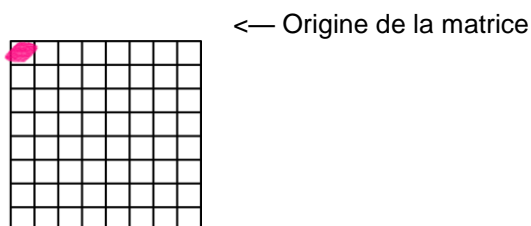
À TOI!

D'après l'énoncé, reproduis sur les quadrillages ci-dessous les 3 situations possibles.

Note : Si tu n'as pas toutes les couleurs, utilise un crayon à papier et ajoute des annotations pour préciser la couleur et la vitesse de clignotement.



Rappel : Dans cette activité, l'origine (1,1) de la matrice est le point suivant.



Partie 2 - Programmation

1

Le mur

Va sur mBlock pour commencer à programmer le premier mur.

CONTINUE

Renseigne l'ensemble des coordonnées où se situe le mur de gauche, il en faut 8. Rédige-les selon la manière de l'exemple : (1,8) etc... Commence par le numéro de la colonne puis de la ligne.

.....
.....
.....

VA SUR MBLOCK

Puis crée le bloc « afficher_mur » comme ci-dessous. Cette fonction nous permettra de demander à ton IZlone d'afficher le mur de gauche en situation départ.



PUIS

Crée un programme pour faire afficher le mur gauche dans la situation de départ. Utilise ces genres de bloc pour ton programme :



Note : Il faut que tu rajoutes la suite.

CONTRÔLE

Une fois ton programme de mur fini, vérifie qu'il s'affiche correctement sur l'écran de ton IZlone. Pour cela, il faut que tu rajoutes le bloc « afficher_mur » dans l'initialisation que tu as commencée lors de la séance 1 :



COMPLÈTE

De la même manière, crée les morceaux de code suivants et complète les.

```
definiir mur_clignotant_orange  
mettre à orange le pixel à la ligne n° et à la colonne n°  
faire clignoter normalement le pixel à la ligne n° et à la colonne n°  
  
definiir mur_clignotant_rouge  
mettre à rouge le pixel à la ligne n° et à la colonne n°  
faire clignoter rapidement le pixel à la ligne n° et à la colonne n°
```

Note 1 : il s'agit toujours d'allumer le mur de gauche mais avec des couleurs différentes et des vitesses de clignotement qui varieront.

Note 2 : pas la peine de tester vos nouveaux murs, concentrons à présent sur la suite de l'activité.

2 Les déplacements de la voiture

CONTINUE

Souviens-toi de l'énoncé de la séance 1, sinon n'hésite pas à le relire, et décris avec le plus de précisions possible les déplacements de la voiture (lignes, colonnes, couleur, vitesse, direction, limite)

.....
.....
.....
.....
.....

À TON AVIS

Quels sont les actionneurs que tu devras utiliser pour faire avancer ou reculer ta voiture?

.....
.....

Qu'est-ce qu'une boucle conditionnelle?

.....
.....

Quel type de boucle conditionnelle serait très utile pour lier nos actionneurs et le déplacement de la voiture?

.....
.....
.....

COME ON

Modifie ton initialisation pour qu'elle ressemble au programme ci-contre. Explique ce qu'il se passe ligne par ligne dans ce programme.

```
Arduino - générer le code  
répéter indéfiniment  
si bouton commencer relâché alors  
mettre colonne_voiture à 6  
mettre ligne_voiture à 4  
afficher_mur  
Afficher_voiture  
si bouton à gauche appuyé alors  
ajouter à colonne_voiture -1  
Afficher_voiture
```

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

TESTE

Essaie ton programme dans mBlock. Tu verras que ta voiture se déplace vers la gauche. En revanche, elle ne peut pas revenir en arrière.

Ajoute la partie qui permet de revenir vers la droite.

Tu pourras constater que ta voiture ne s'arrête pas quand elle touche les murs, et surtout que le mur ne change pas de couleur en fonction de la distance de la voiture avec ce dernier.

Nous verrons cette grosse partie lors de la séance 3!

3

Conserve ton programme

SAUVEGARDE

Sauvegarde sur ta session tes morceaux de programme. Ils te seront utiles pour la séance 3.

C'est la fin de la séance 2. Lors de la séance 3, nous finirons ton programme pour que tu aies un véritable radar de recul sur ton IZlone.

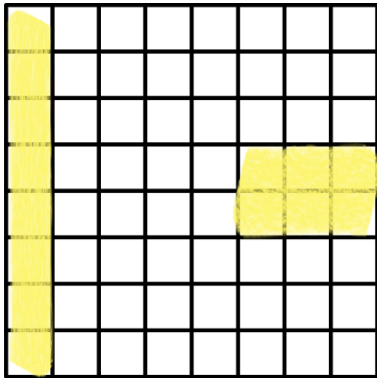
ACTIVITÉ : Apprendre à programmer un radar de recul - Séance 2 - Corrigé

Partie 1 - Comprendre

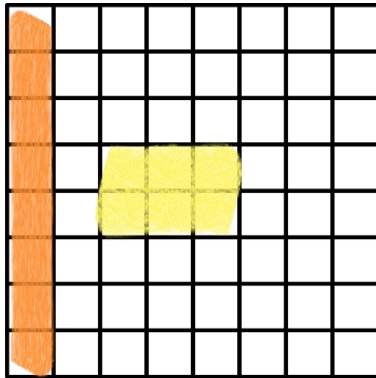
1

Un radar de recul avec IZlone, c'est quoi ?

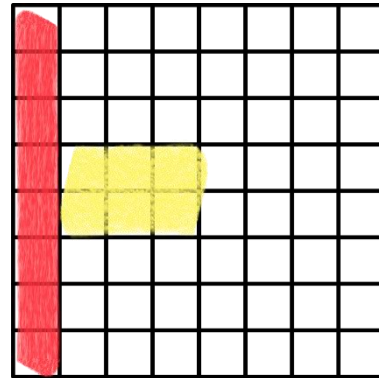
D'après l'énoncé, reproduis sur les quadrillages ci-dessous les 3 situations possibles.



Situation 1 : clignotement lent



Situation 2 : moyen



Situation 3 : rapide

Partie 2 - Programmation

1

Le mur

Renseigne l'ensemble des coordonnées où se situe le mur de gauche, il en faut 8. Rédige-les selon la manière de l'exemple : (1,8) etc... Commence par le numéro de la colonne puis de la ligne.

(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (1,7), (1, 8).

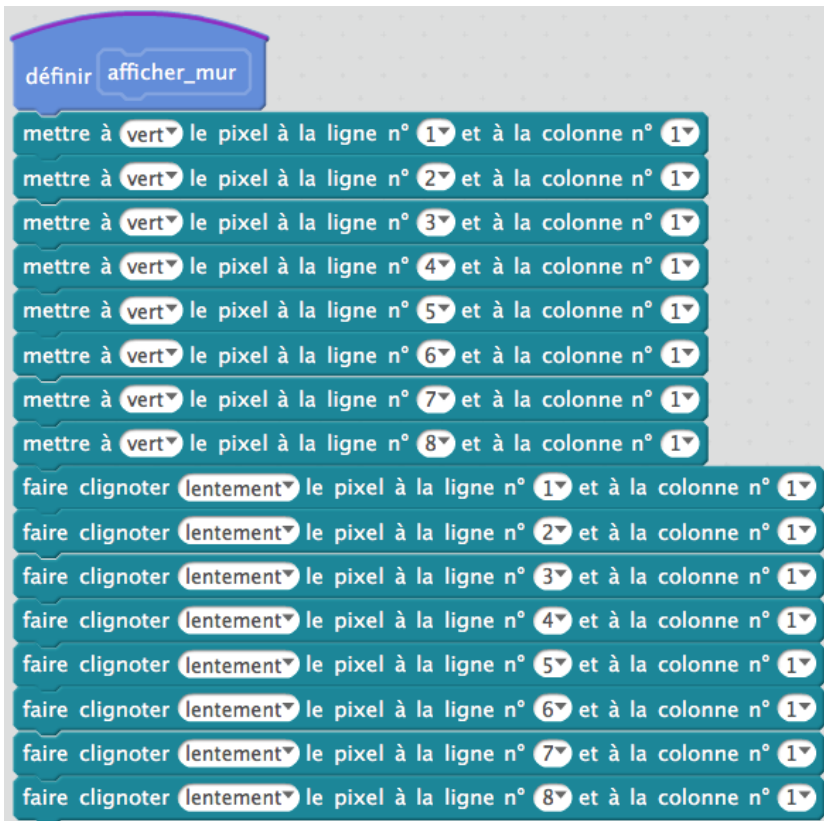
Souvenez-vous de l'origine de la matrice (1, 1) renseignée dans la fiche activité. Plus on va vers la droite de la matrice plus les valeurs des coordonnées augmentent, idem lorsque l'on descend.

Puis crée le bloc « afficher_mur » comme ci-dessous. Cette fonction nous permettra de demander à ton IZlone d'afficher le mur de gauche en situation départ.



Pour créer une variable dans mBlock, allez dans l'onglet Scripts > menu Blocs & variables > cliquez sur "Créer un bloc" (encadré en rouge ci-dessus) > une boîte de dialogue s'ouvre > renseignez le nom du bloc > faites OK > le bloc apparaîtra à l'endroit où vous programmez dans mBlock.

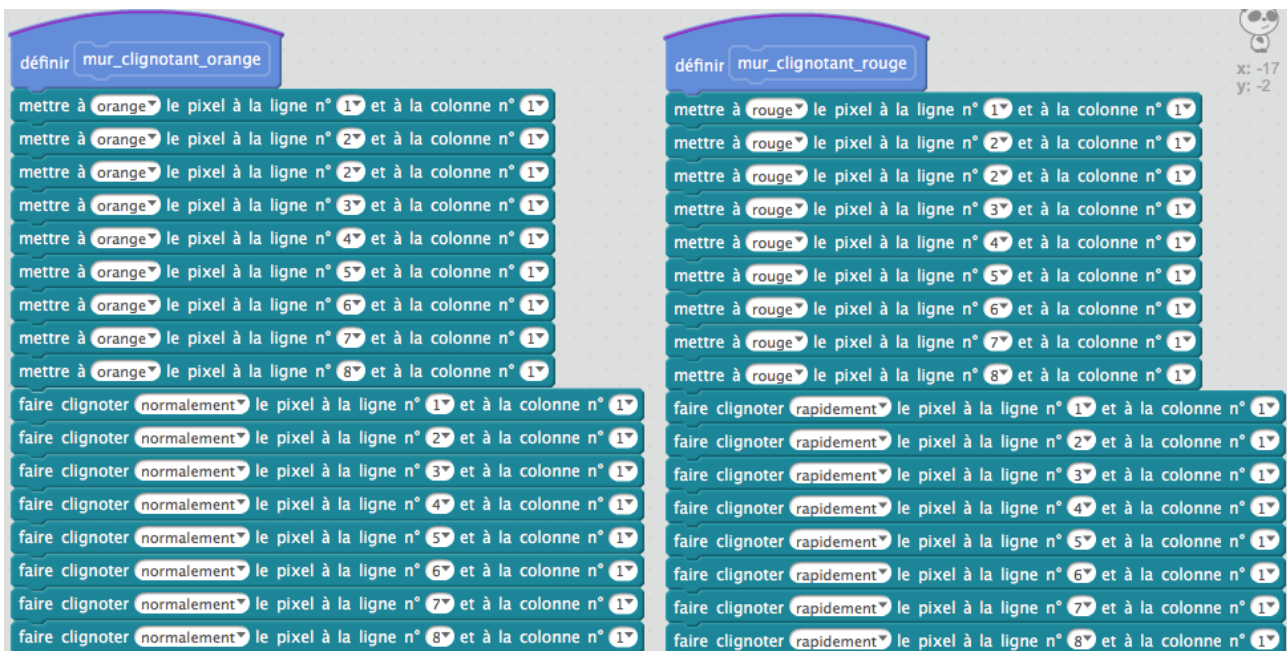
Crée un programme pour faire afficher le mur gauche dans la situation de départ. Utilise ces genres de blocs pour ton programme :



L'ordre dans lequel vous implémenter les blocs dans votre programme n'a pas d'importance. Ici nous avons préféré rassembler les blocs en raison de leurs fonctions.

Ci-dessus nous avons bien demandé à mBlock d'allumer le mur dans sa situation de départ, en vert et clignotant lentement. Les coordonnées sont bien les mêmes que vous avez renseignées précédemment.

De la même manière, crée les morceaux de code suivants et complète-les. (voir l'écran ci-dessous)



Souviens-toi de l'énoncé de la séance 1, sinon n'hésite pas à le relire, et décris avec le plus de précisions possible les déplacements de la voiture (lignes, colonnes, couleur, vitesse, directions, limites)

La voiture ne peut se déplacer uniquement sur les lignes 4 et 5. Quant aux colonnes, elle peut se déplacer sur toutes et même au-delà s'il n'y a pas de limites fixées. La voiture peut sortir de l'écran. La couleur est un élément qui reste vert, qu'importe sa position, seul le mur varie de couleur. La voiture se déplace pixel par pixel. Lorsque j'appuie sur un bouton, elle se déplace d'un pixel vers la droite ou la gauche. Les directions sont vers la gauche ou vers la droite. Une limite évidente sera celle du mur gauche. En effet sur un radar de recul, on s'approche d'un obstacle qu'il vaut mieux ne pas franchir. En revanche pour le côté droit de la matrice, la limite est moins évidente. Vous verrez que nous avons fait le choix de bloquer la voiture selon les limites de la matrice. La voiture sera donc contenue entre les colonnes 1 et 8.

Quels sont les actionneurs que tu devras utiliser pour faire avancer ou reculer ta voiture ?

Les actionneurs que vous utiliserez pour déplacer la voiture sont les boutons de votre IZlone. Et plus particulièrement le bouton gauche et le droit.

Qu'est-ce qu'une boucle conditionnelle ?

Une boucle conditionnelle est quelque chose de fort utile et courant en programmation. Cela permet d'effectuer une action si, et seulement si, une condition est vérifiée.

Quel type de boucle conditionnelle serait très utile pour lier nos actionneurs et le déplacement de la voiture ?

La boucle conditionnelle Si... Alors...

Si j'appuie sur le bouton gauche, Alors la voiture va vers la gauche.

Dans notre cas ici : Si j'appuie sur le bouton gauche, alors j'ajoute à « colonne_voiture » -1.

Je retranche de 1 la valeur stockée dans la variable « colonne_voiture » qui définit la position de la voiture sur la matrice de votre IZlone.

Si vous ne comprenez pas bien cette partie, retournez au corrigé de la séance 1 de cette activité où nous parlions de la construction de la voiture.

Modifie ton initialisation pour qu'elle ressemble au programme ci-contre. Explique ce qu'il se passe ligne par ligne dans ce programme.

- On génère votre code mBlock en un code que votre Arduino peut comprendre.
- On utilise une boucle « répéter indéfiniment » pour que votre programme puisse durer et se répéter une infinité de fois. Sinon vous seriez votre mur et votre voiture s'afficher quelques millisecondes. Cette fonction maintient votre programme affiché sur votre IZlone.
- On place l'initialisation de votre programme dans une boucle Si... Alors... pour que vous puissiez réinitialiser votre programme quand vous voulez. Repartez du début en pressant « commence ».
- On définit les valeurs de doivent prendre vos deux variables pour positionner pour voiture entre les bonnes lignes et aux bonnes colonnes. Souvenez-vous dans votre fonction « afficher_voiture », nous avons créé un bloc de 6 pixels ordonnés. Il manquait simplement la position à laquelle ils allaient apparaître. Reprenez le temps de relire cette partie si nécessaire dans le corrigé de la séance 1.

Dans votre programme principal, on fait appel à vos deux sous-programmes pour qu'ils s'affichent sur votre écran et c'est la fin de la boucle.

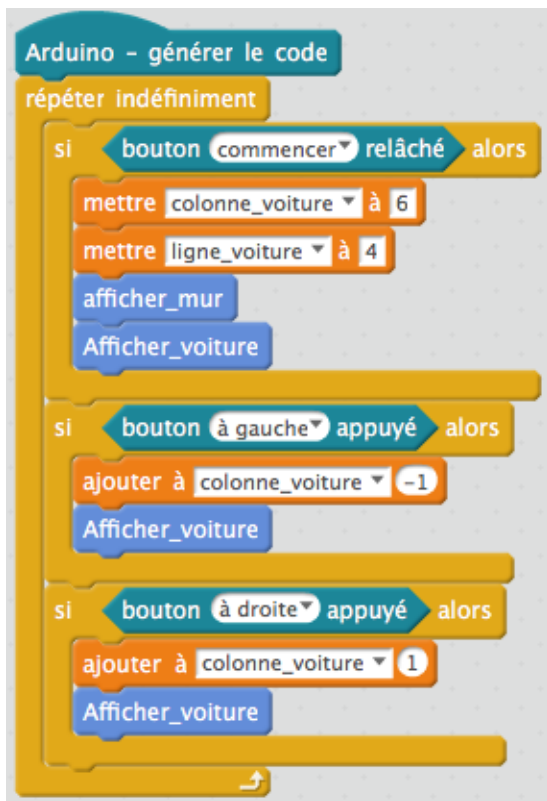
Nouvelle boucle Si... Alors où dès que l'on appuie sur le bouton gauche...

- On retranche à la variable « colonne_voiture » 1 pour faire passer votre voiture de la colonne 6 à la colonne 5.

- On ne touche pas à la variable « ligne_voiture », car on ne veut pas faire monter la voiture sur votre matrice. Elle doit rester entre les lignes 4 et 5.

- On redemande au programme d'afficher la voiture. En effet les coordonnées de la voiture ont été modifiées, mais pas sa position. Il faut la réactualiser. C'est la fin de la boucle.

Essaie ton programme dans mBlock. Tu verras que ta voiture se déplace vers la gauche. En revanche, elle ne peut pas revenir en arrière. Ajoute la partie qui permet de revenir vers la droite.



```
Arduino - générer le code
répéter indéfiniment
  si bouton commencer relâché alors
    mettre colonne_voiture à 6
    mettre ligne_voiture à 4
    afficher_mur
    Afficher_voiture
  si bouton à gauche appuyé alors
    ajouter à colonne_voiture -1
    Afficher_voiture
  si bouton à droite appuyé alors
    ajouter à colonne_voiture 1
    Afficher_voiture
```

ACTIVITÉ : Apprendre à programmer un radar de recul - Séance 3

Extension utilisée : IZlone

Partie 1 - Comprendre

1

Un radar de recul avec IZlone, c'est quoi ?

Avant-propos : Pendant ces séances vous allez programmer un simulateur de radar de recul sur votre IZlone. Chaque séance permet d'apporter un morceau supplémentaire du programme. Il est important de commencer par la séance 1 et de bien suivre l'ordre des séances.

Lors de cette séance 3, nous allons mettre en place le cœur du système de radar de recul. C'est-à-dire pouvoir faire changer la couleur du mur automatiquement en fonction de la proximité de la voiture.

Rappel des situations:

- Situation de départ : la voiture est verte et accolée au bord droit de ton IZlone et le mur clignote lentement en vert.
- Lorsqu'il y a 1 pixel d'écart entre le mur de gauche et la voiture, le mur clignote plus rapidement en orange.
- Lorsque la voiture est au contact du mur de gauche, ce dernier clignote très rapidement en rouge.

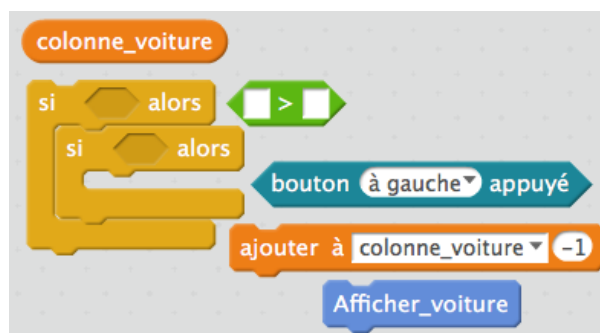
Récupère ton travail réalisé lors de la séance 2.

Lors de la dernière séance, nous étions capables de faire bouger la voiture de droite à gauche. Cependant, la voiture pouvait sortir de l'écran de votre IZlone. Dans la réalité si votre voiture ne peut pas traverser les murs. Il faut donc remédier à ça.

À TOI!

Essayer de retranscrire au **crayon à papier** dans le cadre ci-dessous la phrase suivante en langage de programmation mBlock : « Si j'appuie sur le bouton gauche, la voiture peut aller vers la gauche si elle n'est pas sur la colonne 1 »

Astuce : aide-toi des blocs ci-dessous. Il faut les compléter et les mettre dans le bon ordre.



Partie 2 - Programmation

1

limiter les mouvements à gauche

CONTINUE

Va sur mBlock et retire dans ton code la boucle conditionnelle qui te permettait d'aller vers la gauche et remplace-la par les boucles conditionnelles imbriquées que tu as assemblées à la question précédente. Ainsi tu pourras tester si ta voiture s'arrête bien lorsqu'elle est à la colonne 1. Si tout se passe bien, continue cette activité, sinon retravaille ton code de la question précédente.

2

Changer la couleur du mur

À cet instant, dans l'initialisation de ton programme, tu as demandé l'appel de la fonction « afficher_mur ». Donc au démarrage du programme le mur est bien en vert et clignote lentement.

POURSUIS

La variable « colonne_voiture » doit être égale à quelle valeur pour faire passer le mur à l'orange (retourne lire l'énoncé si tu as un doute)?

.....

Si la variable « colonne_voiture » est égale à la bonne valeur pour passer à l'orange, à quelle fonction faut-il faire appel pour afficher le bon type de mur?

.....

Quel type de boucle te permettrait de rédiger la phrase suivante en langage mBlock? : « Si « colonne_voiture est égale à (ton résultat), alors je fais appel à la fonction « mur_clignotant_orange ».

.....

Rédige en langage mBlock la phrase précédente ci-dessous :

Faut-il les blocs dans le désordre?

CONTINUE

La variable « colonne_voiture » doit être égale à quelle valeur pour faire passer le mur au **rouge**?

.....

Si la variable « colonne_voiture » est égale à la bonne valeur pour passer au rouge, à quelle fonction faut-il faire appel pour afficher le bon type de mur?

« Si la variable « colonne_voiture » est égale à (ton résultat), alors je fais appel à la fonction « mur_clignotant_orange » ».

Rédige en langage mBlock la phrase précédente ci-dessous :

Quand tu es sûr de toi, retranscris tes 2 nouveaux morceaux de code à la suite de la boucle conditionnelle pour aller à gauche dans mBlock

LET'S GO

Que se passe-t-il lorsque la voiture revient à la colonne 4, c'est-à-dire quand la variable « colonne_voiture » = 4? Explique pourquoi.

.....

.....

.....

.....

Pour régler le problème soulevé à la question précédente, complète et arrange ces blocs. Retranscris ton résultat dans le cadre et teste dans mBlock.



Il ne te reste plus qu'à bloquer la voiture pour qu'elle ne sorte pas du mur droit et tu as fini. Reproduis ce que tu as fait pour la bloquer à gauche, mais il faut l'adapter.

3

Conserve ton programme

SAUVEGARDE

Sauvegarde sur ta session ton programme.
Tu as fini l'ensemble de l'activité consacrée au radar de recul.

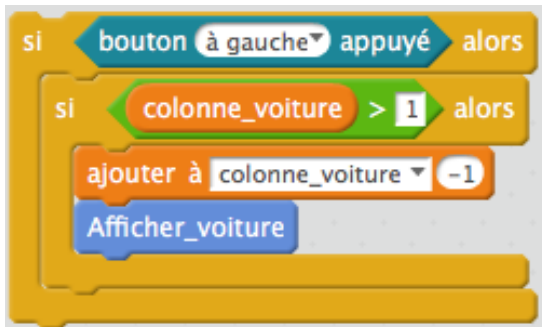
ACTIVITÉ : Apprendre à programmer un radar de recul - Séance 3 - Corrigé

Partie 1 - Comprendre

1

Un radar de recul avec IZlone, c'est quoi ?

Essayer de retranscrire au **crayon à papier** dans le cadre ci-dessous la phrase suivante en langage de programmation mBlock : « Si j'appuie sur le bouton gauche, la voiture peut aller vers la gauche si elle n'est pas sur la colonne 1 »



Ce qui est important ici est de comprendre « Si $colonne_voiture > 1$, Alors la voiture va vers la gauche. »

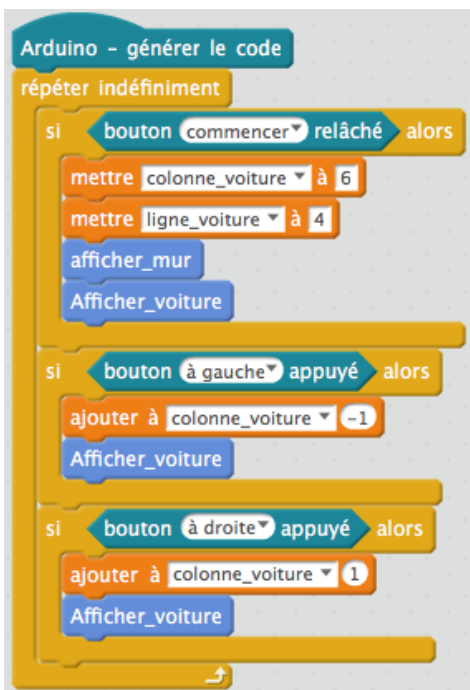
Ce qu'il faut comprendre est que tant que la valeur de la variable « colonne_voiture » est supérieure à 1, la voiture peut aller à gauche. Si cette valeur est égale à 1, alors la voiture ne peut plus aller à la gauche puisque la condition initiale n'est plus vraie. Dans les faits, la voiture sera bloquée à la colonne 1. N'oubliez pas que la valeur de la variable « colonne_voiture » détermine la position de la voiture sur l'axe des abscisses.

Partie 2 - Programmation

1

Limiter les mouvements à gauche

Retirez la partie encadrée en rouge ci-dessous et remplacez-la par les blocs de la question précédente :



2

Changer la couleur du mur

La variable « colonne_voiture » doit être égale à quelle valeur pour faire passer le mur à l'orange (retourne lire l'énoncé si tu as un doute) ?

La variable « colonne_voiture » doit être égale à 3 pour faire passer le mur en orange.

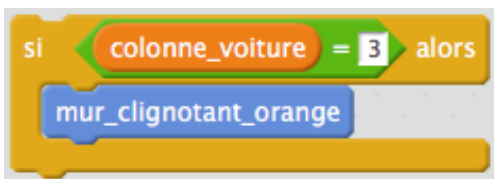
Si la variable « colonne_voiture » est égale à la bonne valeur pour passer à l'orange, à quelle fonction faut-il faire appel pour afficher le bon type de mur ?

Il faut faire appel à la fonction « mur_clignotant_orange ».

Quel type de boucle te permettrait de rédiger la phrase suivante en langage mBlock ? : « Si "colonne_voiture est égale à (ton résultat), alors je fais appel à la fonction 'mur_clignotant_orange'.

Une boucle conditionnelle

Rédige en langage mBlock la phrase précédente ci-dessous :



La variable 'colonne_voiture' doit être égale à quelle valeur pour faire passer le mur au rouge ?

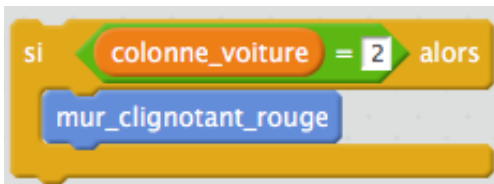
La variable doit être égale à 2

Si la variable 'colonne_voiture' est égale à la bonne valeur pour passer au rouge, à quelle fonction faut-il faire appel pour afficher le bon type de mur ?

La variable 'mur_clignotant_rouge'

'Si la variable 'colonne_voiture' est égale à (ton résultat), alors je fais appel à la fonction 'mur_clignotant_orange'.

Rédige en langage mBlock la phrase précédente ci-dessous :



Quand tu es sûr de toi, retranscris tes 2 nouveaux morceaux de code à la suite de la boucle conditionnelle pour aller à gauche dans mBlock (voir ci-dessous)



Que se passe-t-il lorsque la voiture revient à la colonne 4, c'est-à-dire quand la variable 'colonne_voiture' = 4 ? Explique pourquoi.

Le mur reste orange et clignote à une vitesse normale. Et ce n'est pas normal puisque l'on a quitté la zone de danger correspondant à l'orange. Nous devrions être repassés au vert.

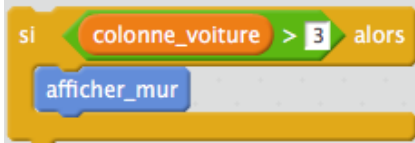
Cependant le programme en l'état fonctionne comme vous lui demandez. Que comprend-il ?

Il sait qu'il faut passer au rouge quand la variable 'colonne_voiture' passe à 2. Il sait aussi passer au orange quand la variable = 3. En l'état quand vous êtes reparti vers la droite pour quitter la zone de danger rouge, vous êtes repassé sur la zone de danger orange. Le mur passe alors à l'orange. Après le programme n'a aucune instruction sur la couleur que doit prendre le mur en 4 et 8. Alors il garde la dernière instruction 'en tête', c'est-à-dire passer le mur au orange.

Il faut donc programmer la chose suivante :

Si 'colonne_voiture' > 3, alors 'afficher_mur'. À cet instant votre mur repassera au vert et correspondra à l'énoncé.

Pour régler le problème soulevé à la question précédente, complète et arrange ces blocs. Retranscris ton résultat dans le cadre et teste dans mBlock.



Il ne te reste plus qu'à bloquer la voiture pour qu'elle ne sorte pas du mur droit et tu as fini. Reproduis ce que tu as fait pour la bloquer à gauche, mais il faut l'adapter.



Remarque : nous avons limité les déplacements de la voiture vers la droite à 6 et non pas à 8 pour l'empêcher de sortir de l'écran.

En effet le repère que la voiture est le pixel le plus en haut à gauche aux coordonnées (6, 4) à la situation de départ. C'est ce pixel qui va être limité à la colonne 6. Ainsi toute la voiture reste à l'écran.

Bien évidemment vous pouvez changer cette instruction comme il vous plaira.

Cette séance est terminée, n'oubliez pas de sauvegarder votre travail.