Stage Android : Infos pratiques et support théorique

Encadrants : Pierre-Olivier Colard et Florian Demesmaeker

01/08/2016 - 05/08/2016

ANDROID

technofutur TIC
centre de compétence

ÉCOLE POLYTECHNIQUE DE LOUVAIN

UCL Université catholique de Louvain

Contents

1 Planning

Bienvenue ! Voici le planning de cette semaine de stage.

1.1 Lundi

Matin Présentation générale, présentation d'Android et d'App Inventor 2. Projet Miaou.
Après-midi Théorie si-sinon et variables. Projet PlusOuMoins. Bonus : mot de passe.

1.2 Mardi

Matin Théorie Horloges. Projet décompte. Théorie procédure. Projet SayMyName.
Après-midi Plusieurs projets : calculatrice, touches de piano.

1.3 Mercredi

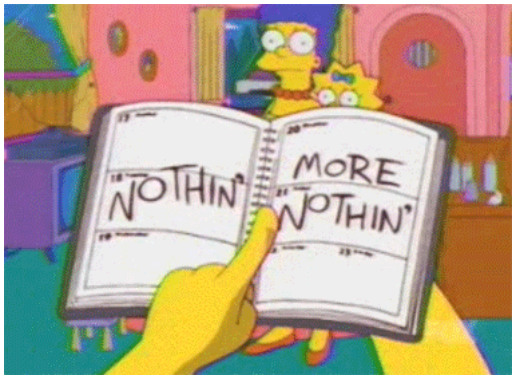
Matin Présentation des images lutins, des animations, des déplacements et des collisions.
Après-midi Projets : Space invaders ou casse-brique. Consignes pour le projet final.

1.4 Jeudi

Matin Conception de votre application **sur papier**.
Après-midi Implémentation de votre application !

1.5 Vendredi

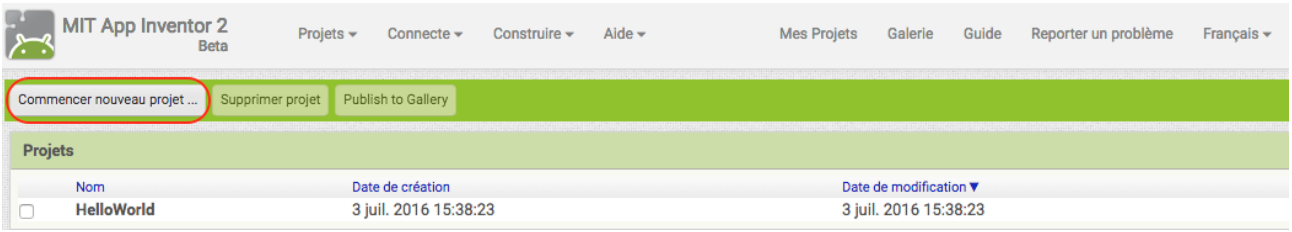
Matin Finalisation de votre application et publication des plus évoluées.
Après-midi Derniers détails de votre application et présentation aux parents.



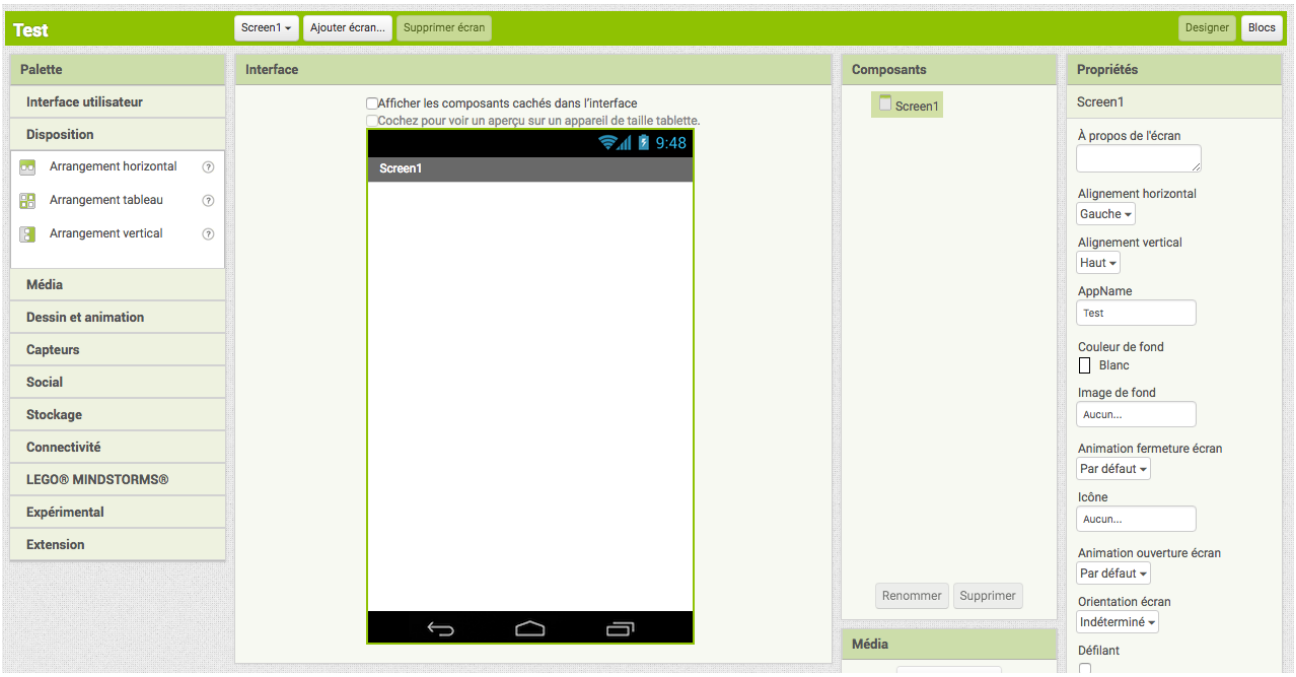
2 Présentation de App Inventor 2

App Inventor est un logiciel développé par le MIT pour simplifier la création d'applications Android. Son utilisation ne nécessite aucune base en programmation. Au lieu de lignes de code, on emboite des blocs pour programmer. Pour y accéder, entrez `http://ai2.appinventor.mit.edu` dans votre navigateur favori.

Pour créer un projet, il suffit d'appuyer sur le bouton `Commencer un nouveau projet...` dans la vue générale d'App Inventor :



Nous arrivons ensuite sur la vue **Designer**:



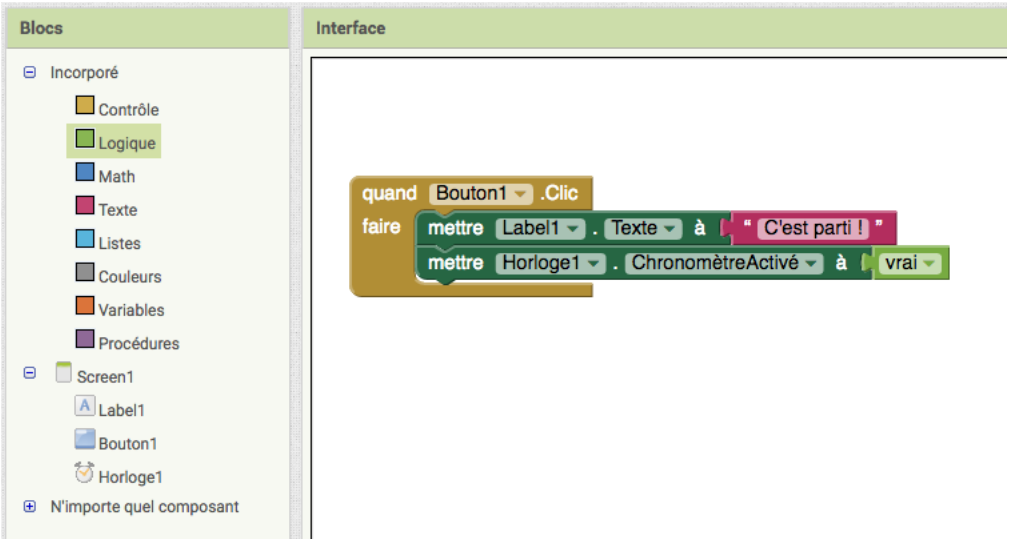
C'est sur cet écran que l'on contruit l'*interface* de notre application. Comment faire ? Nous pouvons chercher les éléments que l'on veut placer sur notre écran dans la Palette et les amener sur l'**Interface**. Par exemple, pour rajouter un bouton sur l'écran, on cherche **Bouton** dans le menu **Interface Utilisateur** de la **Palette**, on clique ensuite sur **Bouton** et sans lâcher le clic, on le déplace jusque dans l'**Interface**.

Nous pouvons placer toute une série d'éléments dans l'**Interface**, comme des boutons, des images et des listes, mais aussi des éléments invisibles qui nous serviront par la suite, comme un minuteur, un son, un objet de reconnaissance vocale,... On peut également organiser tous ces éléments en utilisant des **Arrangements horizontaux**, **Arrangements verticaux** ou des **Arrangements tableaux** disponibles dans le menu **Disposition** de la **Palette**.

Enfin, on observe aussi à droite de l'**Interface** la colonne **Composants** et la colonne **Propriétés**. Dans **Composants**, on retrouve un résumé de tous les objets placés dans l'**Interface**. C'est là qu'on peut changer le nom par lequel on désigne nos objets, par exemple changer *Button1* en *Bouton Start*. Dans **Propriétés**,

on peut changer toutes les caractéristiques d'un objet placé dans l'**Interface**. Par exemple pour l'horloge, on peut déterminer si elle est activée par défaut (**ChronomètreActivé**), si elle s'exécute tout le temps (**TimerAlwaysFire**) et sa longueur (**IntervalleChronomètre**). Pour un **Bouton**, on peut déterminer le texte, la taille, la couleur, etc.

Pour pouvoir programmer, on passe dans la vue des **Blocs** (juste au-dessus de **Propriétés** quand on est dans le **Designer**) :



Dans cette seconde vue, on peut effectuer des actions sur les objets placés dans la vue **Designer**, ou dépendant de ces objets. Par exemple, en cliquant sur **Button1**, le bloc `Quand Button1.Clic faire` apparait, entre autres. Pour l'utiliser, il suffit de le déplacer dans l'**Interface**. Toute une série d'actions (toujours sous forme de blocs) est possible pour chaque objet.

Les actions plus générales sont accessibles depuis le menu **Incorporé** dans les **Blocs**.

- Dans **Contrôle**, on retrouve les boucles, les structures conditionnelles et la gestion des écrans et de l'application en général.
- Dans **Logique**, on retrouve les opérations logiques (vrai, faux, et, ou,...)
- Dans **Math**, on a toutes les opérations mathématiques disponibles.
- Dans **Texte**, on a toute la gestion du texte, telle que concaténer des textes (assembler deux textes en un), vérifier la longueur d'un texte, écrire un texte, etc.
- Dans **Listes**, on a toute la gestion des listes.
- Dans **Couleurs**, on retrouve toutes les couleurs disponibles.
- Dans **Variables**, on a les blocs nécessaires à l'utilisation de variables.
- Dans **Procédures**, on a les blocs nécessaires à la création de fonctions (qu'on n'utilisera a priori pas cette semaine).

3 Support théorique et exemples pratiques

3.1 Les variables

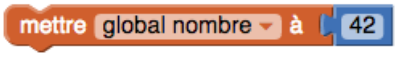
Les variables sont un des concepts de base de l'informatique. Il s'agit de cases mémoires possédant chacune un nom pour la désigner, ainsi qu'un contenu. Le nom d'une variable ne change jamais. Une fois que la variable est déclarée, un nom lui est donné une fois pour toutes. Le contenu de la variable peut quant à lui être régulièrement changé.

Dans App Inventor, les variables peuvent être utilisées grâce à des blocs. Les deux premières choses qu'on doit faire lorsque l'on utilise des variables sont : les déclarer (leur donner un nom) et les initialiser (leur donner un contenu). Dans App Inventor, ces deux actions sont exécutées par un seul bloc:

Ici, on a créé une variable que l'on a appelé *nombre* et qui contient la valeur 1.

On peut utiliser le contenu d'une variable en prenant le bloc orange *obtenir "le nom de la variable"* :

On peut également modifier son contenu en prenant le bloc orange *mettre "le nom de la variable" "le nouveau contenu"* :



Par exemple ici on augmente la valeur de *nombre* de 1. Pouvez-vous expliquer comment on s'y prend ?

3.2 Les structures conditionnelles

Derrière ce nom qui peut sembler un peu barbare à première vue se cache un concept simple : la vérification d'une condition. Une structure conditionnelle est un bloc `si-sinon`, c'est-à-dire un bloc comme celui présenté ci-dessous.

```
Si (condition) alors
    Code 1
Sinon
    Code 2
```

Quand l'ordinateur arrive sur ce bloc, il évalue la condition qui se trouve entre les parenthèses pour décider de ce qu'il va faire par la suite. Si la `condition` est vraie, l'ordinateur exécute les instructions qui sont après le `alors` : le code 1, sinon il exécute les instructions se trouvant après le `sinon` : le code 2.

Ceci nous permet de dire à l'ordinateur que l'on veut qu'il fasse certaines choses en fonction de certaines données. Par exemple, si l'utilisateur a plus de 18 ans, j'aimerais qu'un message "Vous êtes majeur" s'affiche,

tandis que si l'utilisateur a moins de 18 ans, alors j'aimerais qu'un message "Vous êtes mineur" s'affiche. Maintenant que nous avons vu comment utiliser le bloc *si-sinon*, voyons l'exemple que l'on vient de donner.

Sur AppInventor 2, en utilisant un bloc *si-sinon*, qui se trouve sous l'onglet marron **Contrôles**, on a le résultat donné ci-dessous. Petite indication : afin d'ajouter un `sinon`, il suffit de cliquer sur le petit bouton bleu juste à gauche du `si`.

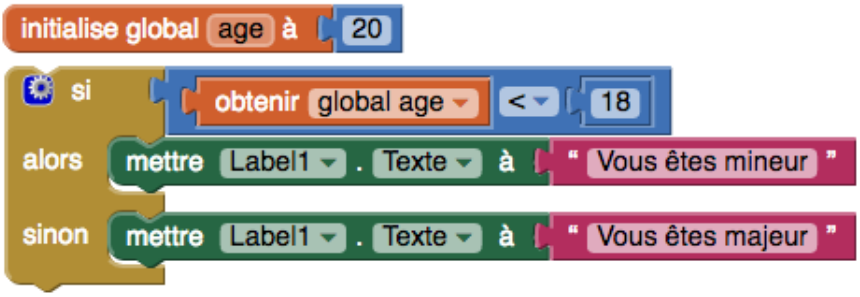


Figure 1: Notre exemple

Les blocs en vert foncé proviennent d'un **label** qu'il faut créer dans le **Designer**, ceux en rose de l'onglet **Texte**, les bleus de l'onglet **Math** et enfin les oranges de l'onglet **Variables**. On a ici utilisé une variable globale `age`, à laquelle on peut donner n'importe quel nombre pour définir l'âge de l'utilisateur. On vérifie ensuite sa valeur dans la condition du `si`, pour finalement afficher un des deux messages.

Ici, puisque l'on a mis la variable `age` à la valeur 20, la condition **ne** sera **pas** vérifiée, et c'est donc les instructions après le `sinon` qui seront exécutées. Il sera alors affiché "Vous êtes majeur" sur le `label1`.

Note Afin de tester ce petit exemple, il faut insérer le bloc *si-sinon* quelque part. Nous suggérons de créer un bouton dans le **Designer**, puis d'insérer notre bloc *si-sinon* dans l'instruction "Quand Button1.Clic ...". On peut trouver cette instruction lorsque l'on clique sous l'onglet Button1 à gauche, dans les **Blocs**.

Pour terminer cette partie, notons qu'il est possible de vérifier autant de conditions que l'on veut dans une structure conditionnelle. On utilise pour cela le même bloc *si-sinon* que celui décrit ci-dessus, auquel on ajoute un ou plusieurs `sinon si`. On associe à `sinon si` une condition, tout comme pour un `si` ! Voyons tout de suite une application concrète : reprenons l'exemple présenté ci-dessus. On veut afficher "Vous êtes mineur" si l'utilisateur a moins de 18 ans ou bien "Vous êtes majeur" si l'utilisateur a strictement plus de 18 ans ou bien "Tout juste majeur !" si l'utilisateur a exactement 18 ans. La solution est donnée ci-dessous. Tout comme pour l'ajout du `sinon`, il suffit de cliquer sur le petit bouton bleu juste à gauche du `si` pour ajouter un `sinon si`.

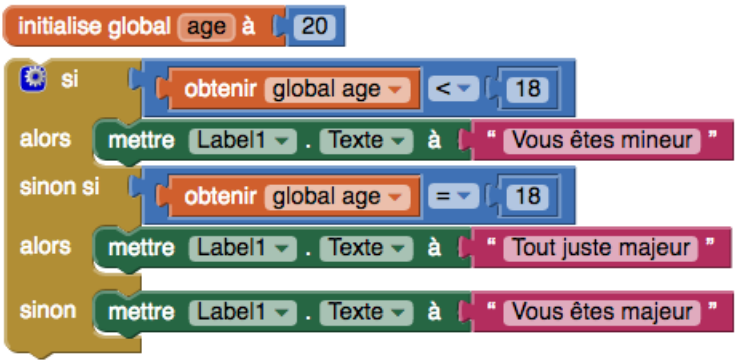
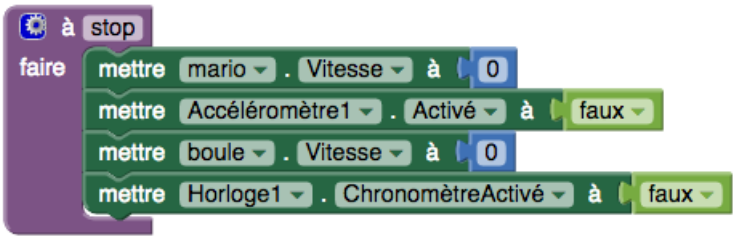


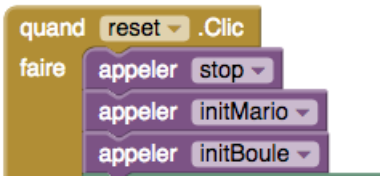
Figure 2: Notre exemple amélioré

3.3 Les procédures

Pour développer votre application, les procédures ne sont pas indispensables mais elles permettent de clarifier et simplifier ce que vous faites. Une procédure est juste une série d'instructions (des blocs). En voici un exemple :



Cette procédure s'appelle `stop` et contient 4 instructions (blocs). Ces 4 instructions sont effectuées chaque fois qu'on **appelle** la procédure `stop`. Dans ce cas-ci, elle permet de figer certains éléments : l'accéléromètre, la vitesse de Mario, ... Par exemple, on pourrait appeler la procédure `stop` lorsque le jeu est perdu ou gagné. Voici un exemple d'appel :



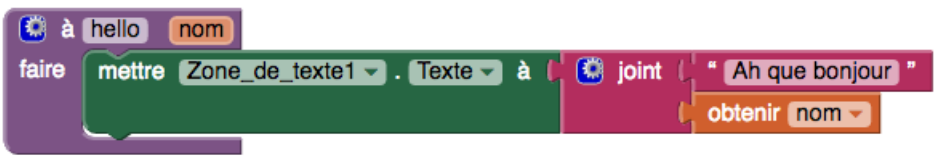
Ici, lorsque le bouton Reset est cliqué, on appelle la procédure `stop`, ce qui veut dire qu'on exécute les 4 instructions qui se trouvent dans `stop`, c'est-à-dire de mettre la vitesse de Mario à 0, d'arrêter l'accéléromètre, ...

Maintenant que l'on sait créer une procédure et l'appeler, nous pouvons rendre notre application plus claire et plus simple. Par exemple si vous avez une série de blocs qui se trouve à plusieurs endroits de votre application, vous pouvez :

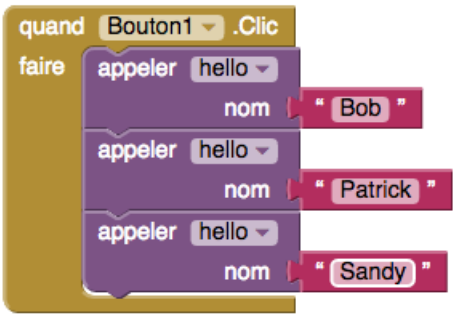
- Créer une nouvelle procédure
- Lui donner un nom
- Y placer la série de blocs
- Retirer la série de blocs qui se trouve à plusieurs endroits dans votre application
- Aux endroits où vous avez enlevé la série de blocs, y placer un **appel** à votre nouvelle procédure

3.3.1 Les procédures avec entrée(s)

Les procédures ont plus d'un tour dans leur sac, nous pouvons aussi leur donner une ou plusieurs **entrée(s)**. Ces entrées peuvent être utilisées par la procédure pour afficher, calculer, ... quelque chose. Voici tout de suite un exemple :



On a définit la procédure `hello` qui prend une **entrée** qu'on appelle ici `nom`. Quand la procédure est appelée, elle change le contenu d'une zone de texte : elle y met *Bonjour* ainsi que le `nom` donné en entrée. En voici un exemple d'utilisation :



Ici on appelle 3 fois la procédure `hello`. Après le premier appel, le contenu de la zone de texte sera *Bonjour Bob*, après le deuxième appel, le contenu sera *Bonjour Patrick* et enfin après le troisème appel, le contenu sera *Bonjour Sandy*.

3.4 Les boucles

Les boucles sont très utilisées en informatique pour répéter une opération. On va répéter cette opération tant qu'une condition n'est pas vérifiée. Par exemple, lorsque l'on doit faire une punition qui consiste à écrire 500 fois "Je ne parlerai pas en cours", il suffit d'utiliser une boucle qui écrit "Je ne parlerai pas en cours" tant que cela n'a pas été fait 500 fois, et qui s'interrompt ensuite.

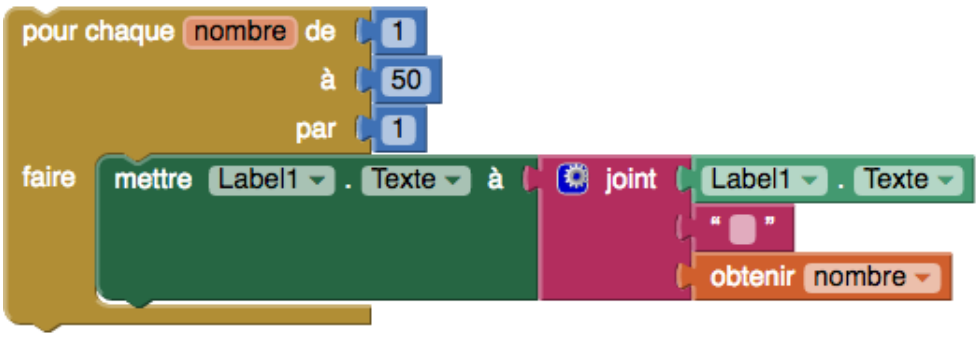
Les 2 boucles les plus utilisées en informatique sont les boucles `tant que` et `pour`. Les boucles `pour` sont en fait un type particulier de boucles `tant que`.

La boucle `tant que` effectue une opération tant qu'une condition n'est pas respectée. Par exemple, si l'on veut jouer un son tant que l'utilisateur n'a pas appuyé sur un bouton, on écrira quelque chose comme :

`tant que` "l'utilisateur n'a pas appuyé sur le bouton", `faire` "jouer le son".

Tant que l'utilisateur n'appuie pas, le son sera joué encore et encore, et une fois que l'utilisateur aura appuyé sur le bouton, on sortira de la boucle et le son arrêtera d'être joué.

La boucle `pour` est presque identique à la boucle `tant que`, sauf qu'elle inclut un compteur. En voici un exemple :



Dans cet exemple, une variable appelée `nombre` dont la valeur initiale est 1, sera incrémentée de 1 (+1) à la fin de chaque exécution du contenu de la boucle (dans le `faire`), jusqu'à ce qu'elle vaille 50. La condition à respecter pour rester dans la boucle est que `nombre` soit compris entre 1 et 50. Lorsque l'on rentre dans la boucle, on aura donc `nombre` initialisé à 1, puis 1 qui sera écrit dans le *label1*. Ensuite, `nombre` sera incrémenté de 1, on aura donc `nombre`=2. La condition étant toujours vérifiée (`nombre` est compris entre 1 et 50), on repassera dans le `faire`, et on va donc afficher 2 à côté de 1 dans le *label1*. Puis à nouveau, `nombre` sera incrémenté. Au final, on obtiendra donc une suite des nombres de 1 à 50 dans notre *label1*. Une fois que `nombre` vaudra 51, la condition ne sera plus respectée, et on va donc sortir de la boucle, sans exécuter l'opération.

3.5 Divers

Cette dernière partie mentionne quelques éléments intéressants à utiliser sur App Inventor 2. N'hésitez pas à les tester par vous-mêmes !

3.5.1 Horloge

L'utilisation d'une `horloge` permet d'effectuer une action à intervalles réguliers. On peut la voir comme une boucle qui s'active à intervalles réguliers. Une `horloge` peut être démarrée ou arrêtée et on peut changer sa fréquence. On utilise un bloc `Quand Horloge.Chronomètre faire` pour définir ce que l'on veut qu'il se passe toutes les 2 secondes par exemple. On peut bien sûr choisir si cela doit arriver toutes les 1, 2, 10 secondes, à volonté. On utilise pour cela l'`IntervalleChronomètre`, disponible dans l'onglet **Designer**. On peut aussi le changer au cours de l'utilisation de l'application.

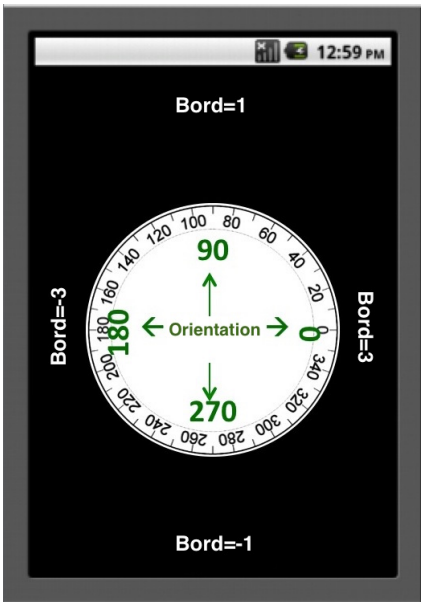
3.5.2 Cadre

Un `cadre` est une zone de travail. Dans cette zone on peut mettre des `balles` et des `Images lutins`. Les premières sont simplement des balles, et les suivantes sont des images que l'on peut *charger* depuis son ordinateur. Ces deux éléments sont présentés après cette section. Un `cadre` permet de faire plusieurs choses : on y place des éléments qui peuvent bouger, on détecte des mouvements de doigts, des touches à l'écran,...

Il est aussi à noter que le système d'axe (X,Y) est un peu particulier. L'origine (0,0) se trouve tout en haut à gauche du `cadre` dans lequel les éléments évoluent. De plus, les Y augmentent en **descendant**. Ce qui veut dire que si votre écran fait 200 pixels de longueur et 100 pixels de largeur, le point tout en *bas à droite* possède les coordonnées (100,200). Ce système de coordonnées est valable pour n'importe quel objet dans votre application !

3.5.3 Balle

Une `balle` est simplement une balle qui peut bouger dans un `cadre`. On peut lui attribuer une vitesse et une orientation, lui permettre de bouger ou la stopper (`Activée`), mais également définir ce qu'elle doit faire lorsqu'on la touche ou qu'on la déplace. Il est aussi possible de définir ce qu'elle doit faire lorsqu'elle touche un bord de l'écran : la faire rebondir par exemple. Il est important de noter que les bords de l'écran possèdent un numéro particulier et que la direction se donne en **degrés**. Voici une image qui nous donne les informations nécessaires :



3.5.4 Image lutin

Une `Image lutin` n'est rien de plus qu'une image que vous fournissez depuis votre ordinateur. On peut définir ce qu'il se passe lorsqu'on la touche ou qu'on la déplace.

Vous en savez désormais assez pour vous lancer dans une application Android avec App Inventor 2, c'est en testant vous-mêmes tout ce qui a été présenté (et bien plus encore) que vous apprendrez le mieux. C'est parti !