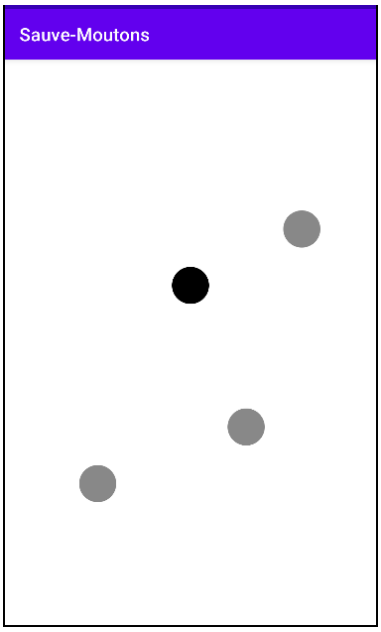
1 Instructions générales et présentation de *Sauve-Moutons*

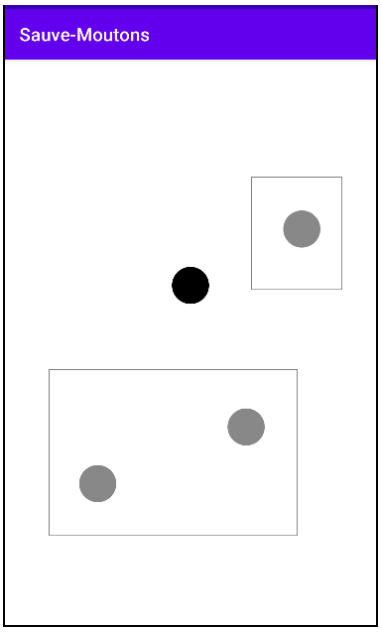
Créez un nouveau projet en langage Java, nommé selon le schéma "TP7_numéro", où "numéro" est votre numéro d'étudiant (sans le 'u' qui le précède).
Si un morceau de votre code ne compile pas, commentez-le avant de déposer votre rendu sur Eprel : un projet qui ne compile pas recevera 0.

Une fois le projet terminé, exportez votre projet (`File` → `Export` → `Export to Zip File`), et rendez ce fichier sur Eprel. N'archivez pas manuellement le dossier contenant votre code !

L'objectif de ce TP est d'implémenter *Sauve-Moutons*, un jeu dans lequel on cherche à construire des enclos autour d'un troupeau de moutons pour les protéger des loups. Une configuration de départ est donnée en Figure 1a : il y a ici trois moutons (en gris), et un loup (en noir). En l'état, le loup est capable de manger les moutons. Pour l'en empêcher, le joueur ou la joueuse devra construire des enclos autour des moutons, comme illustré en Figure 1b.



(a) Un exemple de position de départ : les moutons sont en gris et le loup en noir.



(b) Les moutons sont protégés du loup par des enclos.

FIGURE 1 – Aperçu d'un jeu de *Sauve-Moutons*.

2 Création et affichage du pâturage

Question 1 Créez une classe abstraite `Animal`. Cette classe est destinée à représenter n'importe quel animal (mouton ou loup), avec sa position dans le pâturage et sa couleur. Les attributs de cette classe sont :

— deux entiers `x` et `y`, correspondant à la position, et

— une couleur `couleur` (on rappelle qu'en Java, les couleurs sont représentées par des `int`).
Dotez cette classe d'un constructeur adapté.

Question 2 Créez les classes `Mouton` et `Loup`, qui héritent toutes deux d'`Animal`. Écrivez un constructeur pour chacune de ces classes, qui fera appel au constructeur de la classe `Animal`, de sorte

— qu'un `Mouton` soit gris (on utilisera la constante `Color.GRAY`), et
— qu'un `Loup` soit noir (on utilisera la constante `Color.BLACK`).

Question 3 Créez une classe `Paturage`, qui hérite de `View`, et possède commme attributs

— une liste de `Mouton`, et
— une liste de `Loup`.

On dotera cette classe de son constructeur naturel. Dans le `onCreate()`, on créera un `Paturage` `paturage` (ayant pour l'instant deux listes vides) et on placera `paturage` comme seule `View` de l'activité via `setContentView(paturage)`.

L'attribut `x` d'un `Animal` est un entier compris entre `0` et `100`. Il dénote la position relative de cet `Animal` sur l'axe horizontal : à `0`, il est situé tout à gauche de l'écran, et à `100`, tout à droite. De même, si `y` est nul, alors l'`Animal` est situé tout en haut de l'écran, ou tout en bas si `y` vaut `100`. Ainsi, la position centrale correspond à `(50,50)`.

Ce positionnement relatif permet de créer un `Animal` sans se soucier de la taille du `Paturage`. Ce n'est qu'au moment de dessiner l'`Animal` qu'il faut se préoccuper de calculer ses coordonnées véritables.

Question 4 Complétez la méthode `dessiner()` de `Animal` :

```
public void dessiner(Canvas canvas) {
  int w = canvas.getWidth();
  int h = canvas.getHeight();
  ...
}
```

Cette méthode dessinera sur `canvas` un cercle centré sur la position de l'`Animal`, c'est-à-dire aux coordonnées

`(x/100)*w` et `(y/100)*h`.

Le rayon de ce cercle fera un vingtième du minimum (`Math.min()`) entre `w` et `h`. L'intérieur de ce cercle sera colorié avec la `couleur` de l'`Animal` en question ; il s'agira donc d'un disque gris dans le cas d'un `Mouton`, et noir dans le cas d'un `Loup`.

Question 5 Faites en sorte qu'en se dessinant, un `Paturage` dessine chaque `Animal` qui s'y trouve.

Remplacez le `Paturage` vide de la Question 3 par un `Paturage` avec trois `Moutons` aux coordonnées

$$(27, 75) \qquad (65, 65) \qquad (80, 30)$$

et un `Loup` aux coordonnées

$$(50, 40)\,.$$

Le résultat obtenu devrait ressembler à la Figure 2.

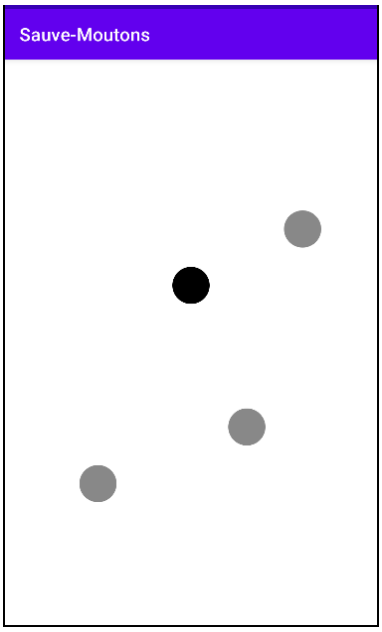


FIGURE 2 – Position de départ des moutons (en gris) et du loup (en noir).

3 Gestion des enclos

Pour séparer nos moutons des loups, on veut pouvoir les protéger dans des enclos rectangulaires.

Question 6 Créez une classe `Enclos` ayant pour attribut quatre entiers `gauche`, `droit`, `haut` et `bas`, qui désigneront respectivement la position réelle du bord gauche, droit, haut et bas du rectangle.

Le constructeur

```
public Enclos(int gaucheOuDroit1, int gaucheOuDroit2, int hautOuBas1, int hautOuBas2) {
  ...
}
```

permettra de créer un `Enclos` où le bord `gauche` sera le plus petit (`Math.min()`) des deux arguments `gaucheOuDroit1` et `gaucheOuDroit2`, et le bord `droit` sera le plus grand (`Math.max()`) de ces deux arguments ; et idem pour `haut` et `bas`.

Question 7 Implémentez la méthode `dessiner()` d'`Enclos`. Comme pour la Question 4, celle-ci attendra un `Canvas` en argument. Cette méthode dessinera les contours du rectangle délimité par `gauche`, `droit`, `haut` et `bas` en noir, avec l'épaisseur de trait par défaut.

On notera que, contrairement à la classe `Animal`, dont les coordonnées sont des pourcentages de la largeur et hauteur du `Paturage`, les coordonnées d'un `Enclos` sont réelles, c'est-à-dire exprimées avec les coordonnées réelles des points de l'écran et non ramenées entre 0 et 100. La différence vient du fait que chaque `Enclos` sera créé par une action de l'utilisateur captée par un `onTouchEvent()`, qui utilise les coordonnées réelles.

Question 8 Ajoutez comme attribut de `Paturage` une liste d'`Enclos` (qui sera vide à la création), et faites en sorte qu'en se dessinant, un `Paturage` dessine tous les `Enclos` qu'il contient.

On veut maintenant que le joueur puisse tracer des enclos sur l'écran.

Question 9 Réécrivez la méthode `onTouchEvent()` de `Paturage` de sorte qu'un enclos soit créé à chaque fois que la joueuse a touché deux fois l'écran.

Plus précisément, si le premier toucher a lieu à la position `(a,b)` et le deuxième toucher a lieu à la position `(c,d)`, alors on souhaite créer (et ajouter au `Paturage`) un `Enclos` dont deux des coins opposés sont situés aux coordonnées `(a,b)` et `(c,d)`. Par exemple, si les deux premiers touchers ont lieu (dans n'importe quel ordre) aux positions marquées en rouge sur la Figure 3, alors on veut obtenir le résultat affiché, c'est-à-dire un `Enclos` délimité par ces deux positions. Les points rouges indiquent juste les positions du doigt, et ne doivent pas être affichés.

Une fois ce premier `Enclos` créé, on veut créer un nouvel `Enclos` qui sera délimité par les troisième et quatrième touchers, puis un troisième `Enclos` qui sera délimité par les cinquième et sixième touchers, etc.

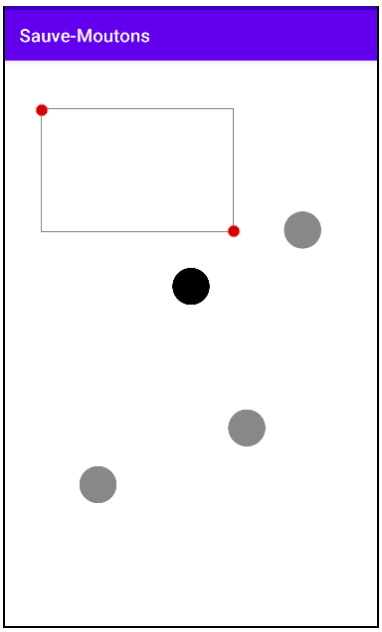


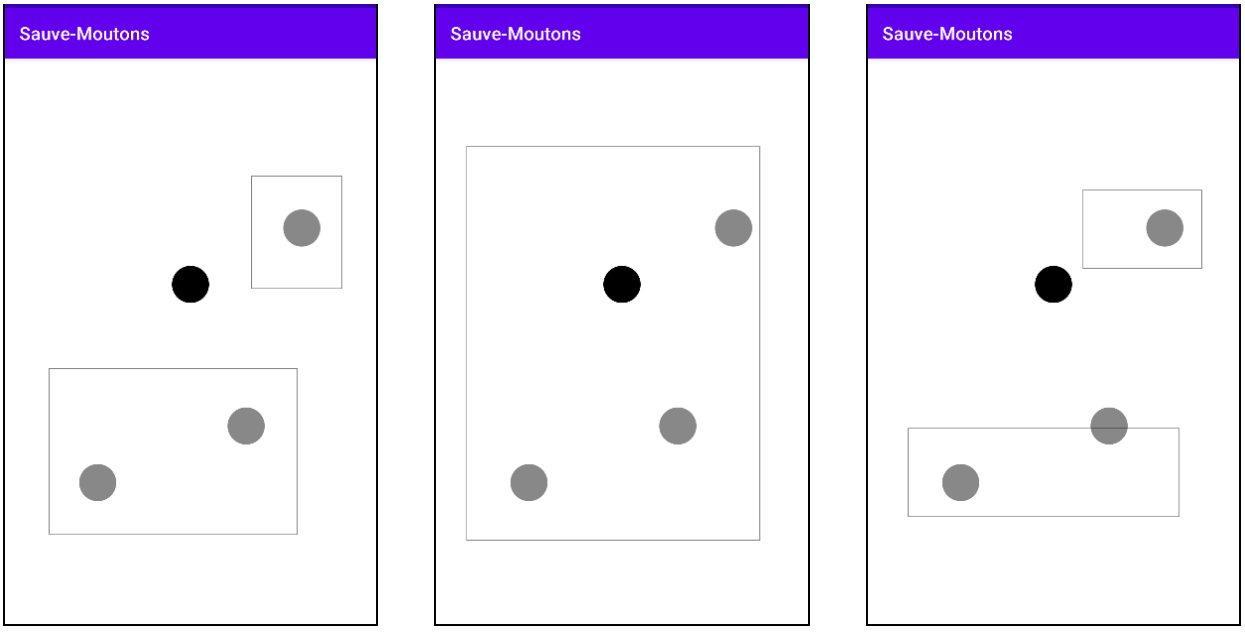
FIGURE 3 – Enclos créé après que le joueur a touché l'écran aux deux endroits indiqués en rouge, dans n'importe quel ordre.

Pour terminer cette première version du jeu, on souhaite savoir si la partie a été remportée ou non. La partie est gagnée si chaque `Mouton` est à l'intérieur d'un `Enclos` et chaque `Loup` est à

l'extérieur de tous les `Enclos`. Si un `Loup` est à l'intérieur d'un `Enclos`, alors on déclarera tout de suite que la partie est perdue. On déclarera aussi une défaite immédiate si un `Enclos` touche un `Mouton` ou un `Loup`.

Ces conditions de victoire sont représentées en Figure 4.

Question 10 Faites en sorte qu'en cas de victoire ou de défaite, un `Toast` affichant "Gagné !" ou "Perdu !" s'affiche en bas de l'écran.



(a) Tous les moutons sont dans un enclos, le loup est au dehors : la partie est **gagnée**.

(b) Le loup est dans un enclos : la partie est **perdue**.

(c) Un animal est à cheval sur un enclos : la partie est **perdue**.

FIGURE 4 – Conditions de victoire et de défaite.

4 Configuration de départ

Nous avons pour l'instant considéré une configuration de départ statique, avec trois moutons et un loup toujours placés au même endroit (cf. Question 5). Pour pimenter le jeu, on veut faire varier le positionnement de départ.

Question 11 La page `https://www.lacl.fr/julien.grange/Enseignements/Programmation_mobile/23_24/TP7/animaux.php` génère un fichier JSON contenant des positions aléatoires pour les moutons et les loups.

Grâce à une `JSONObjectRequest()` de la bibliothèque Volley, utilisez les positions fournies sur cette page pour placer les moutons et les loups. On créera le `Paturage` uniquement à la réception du JSON.

Un exemple d'une telle configuration de départ est donné en Figure 5.

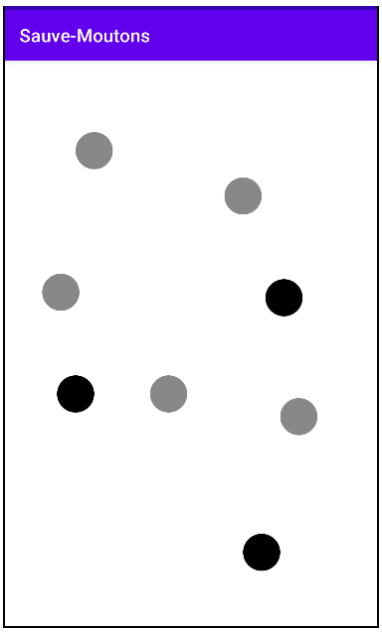


FIGURE 5 – Exemple de configuration de départ spécifiée par le JSON.

5 Enclos glissant

Exportez et rendez sur Eprel une première version de votre projet. Vous pourrez ensuite téléverser votre projet modifié par la Question 12 sous la forme d'un deuxième fichier zip nommé "TP7v2_numéro".

Jusque-là, nous avons créé un `Enclos` pour chaque paire de toucher de l'écran (cf. Question 9). Pour terminer, on aimerait modifier cette manière de faire. Plutôt que de procéder par double-toucher, pour créer un `Enclos`, on aimerait

(i) que le joueur pose son doigt l'écran (cet emplacement détermine l'un des coins de l'`Enclos`),

(ii) fasse glisser son doigt sur l'écran,

(iii) puis retire son doigt de l'écran, ce qui marquera l'emplacement du coin opposé de l'`Enclos`.

De plus, on aimerait que l'`Enclos` en cours de construction soit affiché durant toute la phase (ii). Le rectangle affiché aura donc un coin fixé (celui de l'étape (i)), et le coin opposé suivra le parcours du doigt durant l'étape (ii). Ce n'est que lorsque le doigt aura été relevé que l'`Enclos` sera fixé définitivement.

Question 12 Modifiez la méthode `onTouchEvent()` de la Question 9 pour que la création des `Enclos` se fasse comme indiqué ci-dessus. Attention : si vous n'arrivez pas à répondre à temps à cette question, retournez à la version précédente pour ne pas perdre les points de la Question 9.