Bonus 1: Prise en main Scala, ScalaIDE, Scribble

1 Préliminaires

En vous aidant si besoin de la Notice ScalaIDE :

1. Lancez l'éditeur ScalaIDE (Eclipse 4.7 Scala)
2. Importez le projet ScalaIDE `Bonus1.zip` disponible depuis Moodle.

Pour les deux parties suivantes, pensez à consulter les documentations des bibliothèques standard Scala, Scribble, et du package `fr.istic.si2.math`. Toutes les documentations sont disponibles en ligne (voir page Moodle).

2 Sentinelle : package `fr.istic.si2.bonus1.sentinelle`

On souhaite programmer le jeu graphique très simple suivant : dans une fenêtre graphique, une flèche rouge s'oriente à chaque instant vers le pointeur de la souris. Le jeu ne s'arrête jamais. Initialement, la flèche est par défaut positionnée comme pointant vers la droite (voir la vidéo de démonstration disponible depuis Moodle).

Dans ce package, on vous fournit les bases de l'application de `Sentinelle`. L'objectif est de programmer par vous-même les parties manquantes. Celles-ci sont repérées par une tâche `TODO` dans le fichier `definitions.scala`, et comportent :

— la création de l'image même de la flèche ;

— le calcul de l'angle inscrit entre l'axe des abcisses et un point passé en paramètre ;

— le calcul de l'image de la flèche après avoir subi une rotation.

Exercice 1 (Sentinelle)**.**

— *Programmez toutes les fonctions signalées par une tâche* `TODO` *dans le fichier* `definitions.scala`*.*

— *Songez à tester l'application après chaque étape ! Il vous suffit pour cela d'exécuter l'application située dans le fichier* `main.scala`*.*

— *Indication : pour le calcul de l'angle de rotation en degrés, vous pourrez utiliser la fonction* `scala.math.atan2`*, dont voici la documentation. Attention à l'ordre des paramètres. Egalement, l'angle retourné par* `scala.math.atan2` *est exprimé en radians. Pensez aux fonctions de conversions disponibles dans les librairies standard Scala et* `fr.istic.si2.math` *!*

```
/** Converts rectangular coordinates (x, y) to polar (r, theta).
 *  @param  x the ordinate coordinate
 *  @param  y the abscissa coordinate
 *  @return the "theta" component of the point (r, theta) in polar
 *          coordinates that corresponds to the point (x, y) in
 *          cartesian coordinates.
 */
def atan2(y: Double, x: Double): Double
```

3 A la recherche du pixel caché : package `fr.istic.si2.bonus1.secretPixel`

On souhaite programmer le jeu graphique suivant : un pixel secret se trouve quelque part dans une fenêtre graphique et le but du jeu est de passer suffisamment près avec la souris. Pour y arriver, le jeu donne une indication : il modifie l'opacité (alpha) de la couleur de la fenêtre graphique en fonction de la position du curseur de souris : plus on est près du pixel secret, plus le fond de la fenêtre devient blanc ; plus on en est loin, plus le fond est noir. Initialement, la fenêtre est noire. Regardez la vidéo de démonstration disponible depuis Moodle pour comprendre l'utilisation de l'application.

Fichiers fournis. A vous de compléter les définitions manquantes repérées par marqueurs `TODO` dans le package. Consultez la documentation Scribble à propos du codage RGB des couleurs et l'accès aux champs des couleurs.

Test. Pour tester régulièrement le fonctionnement global de l'application, exécuter l'application `PixelSecret` située dans le fichier `main.scala`. Utiliser la fonction `jouer`, qui déclenche l'interface graphique du jeu, avec un niveau de difficulté indiqué par la val `niveau` de type `String` (`"facile"`, `"moyen"`, ou `"difficile"`). Les niveaux de difficultés sont expliqués plus bas.

Informations supplémentaires. On donne dans la suite quelques indications et explications sur les fonctions fournies et à programmer. Cela explique *le découpage fonctionnel* de l'application. Lisez ce paragraphe attentivement.

`WIDTH, HEIGHT` définissent les dimensions, en pixels, de la fenêtre du jeu.

`SECRETX, SECRETY` Ces définitions déterminent le pixel secret, et sont déterminées au démarrage de l'application, par tirage au hasard, dans les dimensions de la fenêtre.

`distance` Cette fonction calcule la distance euclidienne entre deux points.

`distSecret` Cette fonction calcule la distance entière entre un point donné et le point secret.

`MAXDIST` définit la distance maximale du pixel secret à un des quatre bords de la fenêtre.

`couleurInit` La couleur initiale de la fenêtre graphique.

`jeu` retourne une image de fond pour la fenêtre graphique. La couleur de l'image est passée en paramètre.

`opacite` Cette fonction a pour rôle de ramener la distance entre un point de la fenêtre et le pixel secret à l'intervalle d'entiers $[0; 255]$, dans le but de faire un lien entre distance et opacité. Plus la distance est proche de `MAXDIST`, plus le résultat doit être proche de 255. Par exemple, pour un point correspondant au point secret, la valeur calculée doit être 0. La variation de l'opacité devra être proportionnelle à la distance au point secret. Cette fonction permet de définir ensuite une fonction `calculCouleur`, calculant la couleur opacifiée correctement.

`jeuFini` Cette fonction détermine si le jeu est fini. C'est à vous de choisir, pour chaque niveau, une valeur adaptée de l'opacité en dessous de laquelle le jeu se termine. Plus l'opacité est faible, plus la souris est proche du pixel secret, et inversement. Une limite élevée (proche de 255) signifie donc que le rayon de tolérance autour du pixel secret est grand (niveau facile). Une limite basse (proche de 0) signifie qu'il faut passer très près du pixel secret pour gagner.

`finDuJeu` Cette fonction construit la fenêtre graphique affichée quand le jeu est terminé. La couleur courante de la fenêtre est aussi passée en paramètre. Par exemple, en fin de jeu, la fenêtre graphique devrait indiquer au joueur la position du pixel secret, ainsi qu'un message de félicitations.