Problem Solving pour le cours d'I.C.S.

February 10, 2016

Langage: Etudiant niveau intermédiaire, plus lambda;

Dans le premier tutoriel, vous avez vu comment accéder à une bibliothèque qui contient des fonctions pour manipuler des images, par l'expression:

```
(require 2htdp/image)
```

La documentation de cette bibliothèque est dans la documentation Racket -> How To Design Programs Teachpacks (Comment concevoir des programmes Teachpacks) -> 2.3 "image.rkt".

Le fichier `checkerboard.rkt` qui vous avait été envoyé par mail, avait des exemples d'utilisation de certaines fonctions de cette bibliothèque, incluant `square` et `circle` pour dessiner des figures géométriques simples, et `beside` et `above` pour combiner des images.

Dans ce "problem-solving", vous allez les utiliser ainsi que d'autres fonctions de cette bibliothèque pour créer de belles et intéressantes images inhabituelles ou non. Votre code doit satisfaire aux exigences suivantes:

- **Au moins une fonction qui produit une image doit utiliser la récursivité (structurelle préférablement) des nombres naturels non bornés, ou des listes non bornées, ou les deux.**
- Pour chaque fonction, votre code doit contenir un "purpose" (le but) et un contrat. Le type d'une image est `Image`.
- Pour les fonctions qui produisent des images: à la place de faire des tests, votre code doit contenir des expressions qui illustrent son utilisation en affichant des exemples d'images dans la fenêtre d'intéraction lorsque le programme est exécuté. Chaque expression d'une évaluation d'image doit être précédée par une phase qui décrit ce que crée l'image. Par exemple:

```
"Voici le résultat de (circle 20 'solid 'red)."
(circle 20 'solid 'red)
```

- Pour des fonctions qui produisent des nombres, des chaînes de caractères ("strings"), des symboles, des booléens, ou des listes et structures qui les contiennent, vous devez fournir des exemples et des tests comme c'est le cas dans la recette de conception d'un programme ("design recipe").

Vous pouvez rechercher des fonctions dans la bibliothèque d'images dans la documentation Racket, mais pour vous éviter tous les lire, en voici un certain nombre que vous trouverez probablement utile (il y'a n'a beaucoup d'autres), vous trouverez ci-dessous des exemples de descriptions:
Examples:

```
> (square 20 'solid 'red)
```

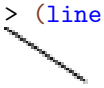
```
> (rectangle 20 30 'outline 'black)
```

```
> (rectangle 20 30 'outline (pen 'black 4 'solid 'round 'round))
```

```
> (triangle 20 'solid (color 0 0 255))
```

```
> (ellipse 20 30 'outline 'green)
```

```
> (line 30 20 'black)
```



```
> (star-polygon 40 5 2 'solid 'seagreen)
```

```
> (radial-star 8 8 32 "solid" "darkslategray")
```

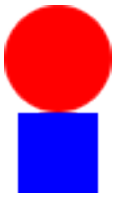
```
> (regular-polygon 10 8 'solid 'purple)
```

```
> (overlay (rectangle 10 40 'solid 'orange)
           (ellipse 40 10 'solid 'purple)
           (circle 17 'solid 'red))
```

```
> (overlay/offset (rectangle 10 30 'solid 'orange)
                  5 15
                  (ellipse 30 10 'solid 'purple))
```

```
> (beside (circle 20 'solid 'red) (square 30 'solid 'blue))
```

```
> (above (circle 20 'solid 'red) (square 30 'solid 'blue))
```



```
> (rotate 45 (ellipse 40 10 'solid 'purple))
```

```
> (scale/xy 1 1/2 (flip-vertical (star 40 'solid 'green)))
```

Les désignations des couleurs prédéfinies peuvent être trouvées dans la documentation Racket -> The Racket Drawing Toolkit -> 7 Color Database.

Enfin, si vous constatez que vos images prennent beaucoup de temps pour ce dessiner, la fonction `freeze` peut vous aider. En effet, au cours d'une exécution, les images sont représentées comme une séquence d'actions qui sont uniquement effectuées si l'image doit être affichée. Si vous utilisez plusieurs fois une sous-image, elle sera redessinée à chaque fois, dans chaque cas vous povez utiliser `freeze` pour le dessiner une fois et ensuite utiliser le dessin (l'image bitmap) copié (freeze veut dire geler dans le sens capture) Ceci est similaire avec l'utilisation de `local` pour éviter de reévaluer des expressions dont les valeurs sont utilisées plusieurs fois.

Pour soumettre faire: `ICS.rkt`