Université de la Rochelle
INFO-12605C - IHM - TD10 : Transformations 2D

© B. Besserer, R. Péteri

Année universitaire 2017-2018

1 Ergonomie

1.1 Menu circulaire

Ci-contre un exemple de menu circulaire. Lors de l'apparition d'un menu circulaire, le curseur de la souris est habituellement placé dans un espace vide au centre du menu circulaire. L'utilisateur peut alors :
- déplacer le curseur vers l'option de son choix, puis cliquer pour activer l'option visée ;
- ou cliquer au centre du menu pour sortir du menu.

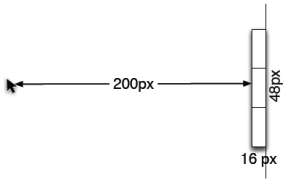
Donner les les avantages / inconvénients du menu circulaire (par exemple au regard de la loi de Fits) ; Préciser pour quel usage les menus circulaires peuvent être avantageux. Que pensez-vous des menus circulaires au regard des diverses modalités de saisies (souris, touchpad / écran tactile, stylet, clavier, ...)

Eléments de correction : *Dans le cas d'un menu d'un menu contextuel standard, les choix proposés sont affichés les uns au-dessus des autres, l'accessibilité de chaque article de menu n'est pas identique, et quelquefois le pointeur, et quittant le menu, fait disparaitre le menu contextuel (ca il n'y a pas d'action specifique pour fermer le menu).*

Dans un environnement graphique, un menu circulaire (appelé en anglais, pie menu, radial menu ou marking menu) est un menu contextuel où les choix disponibles sont affichés de façon circulaire autour du curseur de la souris plutôt que d'être affichés les uns au-dessus des autres. L'accessibilité est identique quelque soit l'article à selectionner.
La sélection d'une option dans un menu circulaire est très facile avec un stylet et assez facile avec une souris. La sélection d'une option est aussi très facile avec des raccourcis claviers utilisant les touches directionnelles (les 4 flèches) ou les touches du pavé numérique, surtout pour les menus de 4 ou 8 éléments. La sélection d'une option dans un menu circulaire peut conduire à un autre menu circulaire.

1.2 Loi de Fitts

On place une barre d'outils (similaire au "dock" Mac OSX) mais à droite de l'écran, accolé au bord droit de l'écran. Les icones de cette barre d'outils ont une hauteur de 48 pixels et une largeur de 16 pixels. Le pointeur de la souris se trouve à 200 pixels de cette barre d'outil. Des mesures ont donné les valeurs des constantes a = 1030 et b = 96. Quel temps faut-il pour atteindre une icône de cette barre d'outil (poser proprement l'équation suffit, pas besoin de faire l'application numérique)



$$T = a + b \log_2\left(2 \frac{D}{W}\right)$$

(Temps : T ; Coefficients : a, b ; Distance : D ; Largeur : W)

Elements de correction : *La loi de Fitts à été donné en cours, quelque fois la formule est différente ... $a + b.log_2(D/W + 1)$. W correspond à la taille du bouton, impliquant sa largeur et sa hauteur (donc sa surface). Plus la surface est grande, plus le temps pour cliquer va diminuer. Mais si la barre d'outils se trouve à droite* ***et que le mouvement du pointeur est arrêté par le bord de***

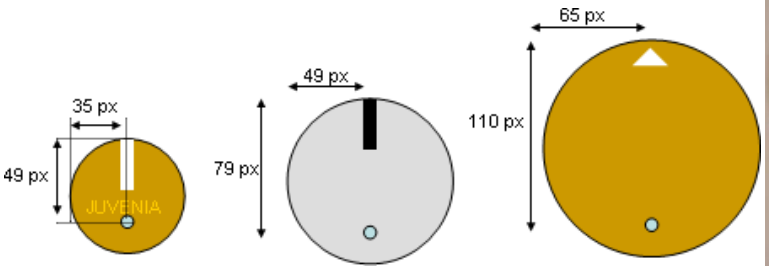
l'écran, alors la largeur du bouton (16pix) n'a pas d'importance seule la hauteur intervient dans le calcul (48pix)... évidemment les paramètres a et b doivent être ajustés dans ce cas.

2 Transformation géométrique suite (préparation au TP)

2.1 Application : JUVENIA 1960

Les montres anciennes révèlent bien de surprises... cette très rare Juvenia des années 1960 utilise 3 disques de taille différentes, affichant les secondes, les minutes et les heures (respectivement du plus petit disque au plus grand), et chaque disque tourne autour d'un axe excentré. On veux reproduire cet affichage.

On dispose de 4 bitmaps et d'un panel, qui recevra le quatrième bitmap (cadran) comme fond (background image).



Dans le constructeur de la Form :

```
m_HourBitmap = new Bitmap("juvenia_hours.png");
m_MinuteBitmap = new Bitmap("juvenia_minutes.png");
m_SecondBitmap = new Bitmap("juvenia_seconds.png");
panel1.BackgroundImage = new Bitmap("juvenia_cadran.png");
```

On met en place un timer générant 1 tick par seconde. Pour que la montre soit de suite et toujours à l'heure, on va, chaque fois que le timer (qui n'aura d'utilité que pour cadencer l'affichage) est écoulé, invoquer l'instruction :

```
DateTime date = DateTime.Now;
```

On peut ensuite accèder aux heures, minutes et seconde que l'on mettra dans des entiers, par exemple :

```
int seconds = date.Second;
```

Il faut maintenant tracer chaque aiguille dans sa bonne position. Exceptionnellement, on placera le code de dessin dans la méthode de gestion de l'événement Tick et non dans la méthode en réaction à l'événement Paint. Si la fenêtre est cachée puis redevient visible, il faudra attendre le prochain Tick pour que l'affichage soit rétabli.

1. Lors de la récupération des données de l'objet DateTime, comment garantir que les heures soient sur 12 et non sur 24 heures ?

   ```
   int hour = date.Hour % 12;
   ```

2. Dans quel odre faut-il dessiner les bitmaps dans le contexte graphique ?
   Chaque opération de dessin se superpose aux données déjà présente dans la surface de dessin du contexte graphique. Il faut donc a chaque fois redessiner, dans l'ordre, le grand disque (heures) le moyen puis le petit (secondes)
3. En supposant que l'origine de la rotation pour chaque bitmap soit l'axe de rotation excentré du disque, donnez la formule permettant directement d'afficher les "aiguilles" dans le bonne position. On rappelle que l'angle donné comme paramètre à RotateTransform est en dégrées, positif dans le sens horaire.

```
gfx.RotateTransform(((float)hour * 30.0F) + ((float)minutes / 2.0F));
gfx.RotateTransform(((float)minutes * 6.0F) + ((float)seconds / 10.0F));
gfx.RotateTransform(((float)seconds * 6.0F));
```

4. Pour chaque "aiguille", donnez la suite des instructions graphiques permettant de positionner les aiguilles (orientés dans la bonne position) dans le cadran (en superposant les les axes de rotation).

C'est le plus compliqué et il faut procéder avec méthode :
On sait que la suite d'instruction :

```
gfx.RotateTransform(((float)seconds * 6.0F));
gfx.DrawImage(m_SecondBitmap, -35, -49);
```

effectue la rotation de la Bitmap autour du point specifié (ici 35,79), et lors du rendu graphique, ce point de rotation est ramené au point d'origine du contexte graphique.
Si le point de rotation doit etre placé à un endroit précis de la bitmap, il faut faire précéder le code par :

```
gfx.TranslateTransform(xcenter, ycenter);
```

si xcenter, ycenter sont les coordonnées correspondantes au point commun de tous les axes de rotation, sur le fond d'écran.

```
DateTime date1 = DateTime.Now;
int seconds = date1.Second;
int minutes = date1.Minute;
int hour = date1.Hour % 12;

int xcenter = panel1.Width / 2;
int ycenter = panel1.Height / 2;
Graphics gfx;

gfx = panel1.CreateGraphics();
gfx.InterpolationMode = System.Drawing.Drawing2D.InterpolationMode.HighQualityBicubic;

gfx.TranslateTransform(xcenter, ycenter);
gfx.RotateTransform(((float)hour * 30.0F) + ((float)minutes / 2.0F));
gfx.DrawImage(m_HourBitmap, -65, -110);


gfx.ResetTransform();
gfx.TranslateTransform(xcenter, ycenter);
gfx.RotateTransform(((float)minutes * 6.0F) + ((float)seconds / 10.0F ));
gfx.DrawImage(m_MinuteBitmap, -49, -79);

gfx.ResetTransform();
gfx.TranslateTransform(xcenter, ycenter);  // pour positionner le dessin au bon endroit
gfx.RotateTransform(((float)seconds * 6.0F));
gfx.DrawImage(m_SecondBitmap, -35, -49);
```