Polytech Annecy-Chambéry – Université Savoie Mont-Blanc

IDU 3 – INFO501

CAULLIREAU Dorian

Date du TP2 (18/12/2020)

Cahier de TP2 : Représentation visuelle d'informations

---

1) L'objectif du TP

L'objectif de ce TP est de représenter des informations issues de capteurs sous la forme d'indicateurs visuels permettant de renseigner très simplement et rapidement sur une situation.

L'application consistera à associer des émoticônes à différents capteurs disponibles sur le site d'Annecy par l'intermédiaire d'une connexion internet.

Il s'agit de capteurs de température dans la salle des serveurs et sur le toit et d'un capteur mesurant la puissance fournie par un onduleur.

Nous avons crée des émoticônes colorées. Une situation totalement normale sera représentée par une émoticône verte, souriante, tandis qu'une situation totalement anormale sera représentée par une émoticône rouge faisant la grimace. Pour les situations intermédiaires la couleur et l'amplitude du sourire ou de la grimace dépendront de la mesure.

Le résultat des émoticons doit être semblable au suivant :

2) Le déroulement du TP

Pour le déroulement du TP, le projet s'est coupé en plusieurs étapes, l'ensemble de ces étapes constitue des fonctionnalités de notre projet.

1. Utilisation et compréhension de pygame
2. Gestion et affichage de l'émoticon
3. Gestion et affichage des boutons
4. Gestion et affichage des capteurs
5. Mise en œuvre de l'application, gestion globale des affichages et des intéractions

3) Réalisation et résultats

Partie 1 : Utilisation de pygame

Q1.b) Les attributs de screen

L'attribut screen permet une récupération des attributs de largeur et de hauteur par les paramètres suivants : get_width() et get_height().

Nous pouvons récupérer ces paramètres de la façon suivante :

```
print(generalConfiguration.screen.get_width())
print(generalConfiguration.screen.get_height())
```

Q1.c) Ajout d'un cadre blanc

D'après la documentation de pygame nous avons la méthode rect() :

Avec les paramètres suivants :

- Self.screen correspondant à la taille de l'écran
- [255,255,255] qui correspond à la couleur RGB du trait
- [19,19,162,162] qui correspond respectivement à la position (x,y) du coins haut gauche, ainsi que la longeur et la largeur du rectangle.

```
pygame.draw.rect(self.screen, [255,255,255], [19,19,162,162],1)
```

Résultat de la fonction rectangle :

Q1.d) Les getters

La création de l'ensemble des getters se fait de la façon suivante :

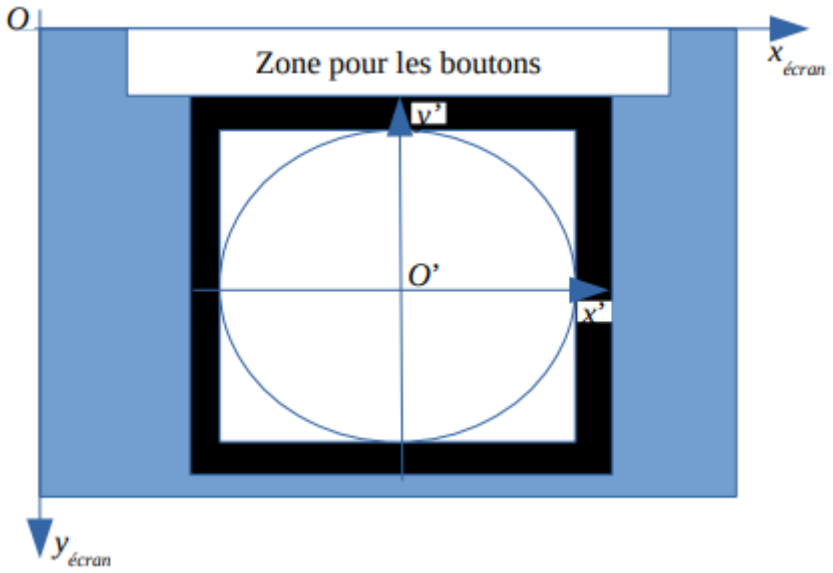
```
def getScreen(self):
      return self.screen
```

Cette méthode est semblable pour la récupération de l'ensemble des attributs suivant : button_width, button_height, emoticon_size, emoticon_border.

L'ensemble des getters pourront alors être utilisés dans les différentes classes qui utilisent GeneralConfiguration. Cela évite de passer directement par les attributs de la classe.

Q2.b) La méthode headToArea

La méthode headToArea() permet le changement de repère, donc le changement des coordonées du plan. La méthode retourne alors une nouvelle position [x,y] suivant le modèle suivant :



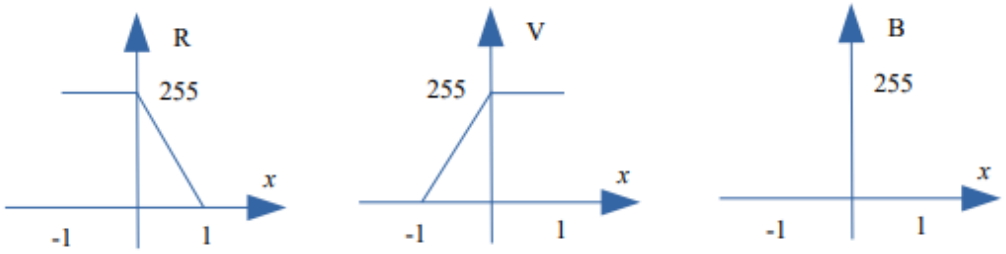
Le fonctionnement de la méthode est le suivant :

```
def headToArea(self, position):

        # x => represente la taille au centre de l'écran, plus la position x
        passer en parametre

        x = int(self.generalConfiguration.screen.get_width()/2 + position[0])

        # y => represente le centre de l'émoticon, en ajoutant la taille des
        boutons du haut, avec le bord de l'émoticon, ainsi que la position x
        passer en parametre

        y = int(self.generalConfiguration.getemoticonSize()/2 +
        self.generalConfiguration.getbuttonHeight() +
        self.generalConfiguration.getemoticonBoarder() + position[1])

        return [x,y]
```

Elle prend en paramètre en position qui permet la retranscription dans le nouveau repère.

Q2.c) La méthode color

La méthode color(x) permet de retourner une couleur associée au paramètre x, suivant le modèle suivant :



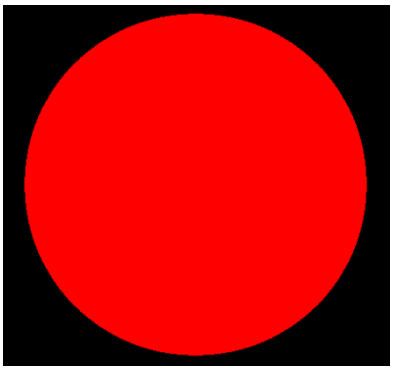
Exemple pour color(1) -> [0,255,0]

Cela permet la gestion de la couleur en fonction de « l'humeur » de l'émoticon.

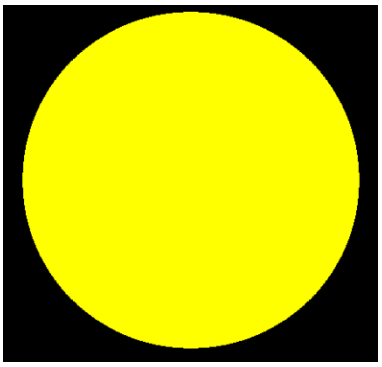
La méthode correspondante au modèle est la suivante :

```
def color(self,color_x):
      if color_x <= 0:
            return [255,int((color_x+1)*255),0]
      else:
            return [int((1-color_x)*255),255,0]
```

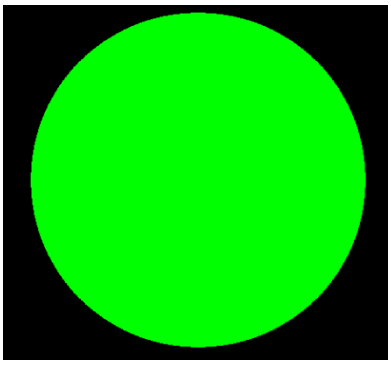
Exemple avec la fonction color en fonction des différents paramètres :



Exemple avec x = -1

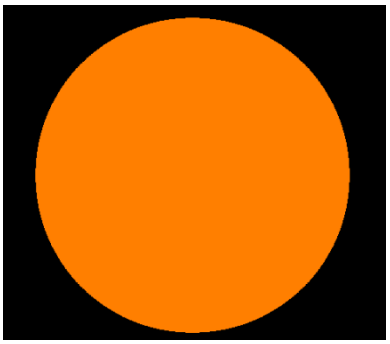


Exemple avec x = 0

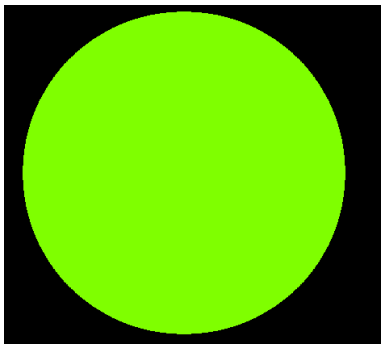


Exemple avec x = 1

Cépendant nous pouvons voir que les couleurs être nuancées, c'est-à-dire par exemple entre le rouge et le jaune.



Exemple avec x = -0.5



Exemple avec x = 0.

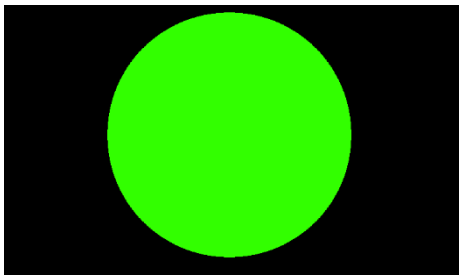
Q2.d) La méthode head

La fonction head() dessine la tête colorée de l'émoticon, elle prend en paramètre un x pour la gestion de la couleur de l'émoticon.

```
def head(self, color_x):

      #Recupération de la taille de l'émoticon
      size = int(self.generalConfiguration.getemoticonSize())

      #Transformation avec la fonction headToArea(), ici centrée donc [0,0]
      pos = self.headToArea([0,0])

      #Dessin du cercle correspondant a la tête
      pygame.draw.circle(self.generalConfiguration.getScreen(),
      self.color(color_x), [pos[0], pos[1]], int(size/2))
```

Avant la création du cercle, nous changeons de référentiel du repère avec la fonction headToArea().

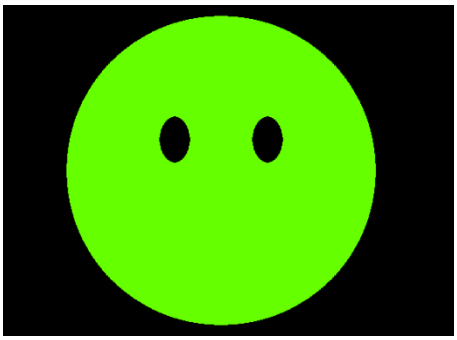
Résultat :



Q2.e) La methode eye

La méthode eye permet l'affichage des yeux de l'émoticon, elle est appelée deux fois pour les deux yeux. Comme précédemment avec la fonction head nous changeons de référentiel pour le repère, avant la création de l'élipse. Les tailles des yeux sont en fonction des paramètres de setEmoticonParameters().

Le résultat attendu est le suivant :

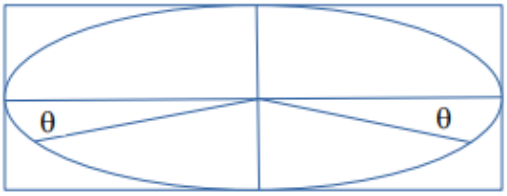


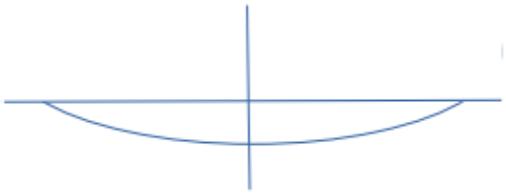
```
def eye(self, position):

      #Gestion des [x,y] pour la position
      eye_emoticone = [position[0]-self.eyeWidth/2, -position[1]-
      self.eyeHeight/2]

      #Transformation avec la fonction headToArea()
      eye_screen = self.headToArea(eye_emoticone)

      #Dessin des ellipse correspondant aux yeux
      pygame.draw.ellipse(self.generalConfiguration.getScreen(), [0,0,0],
      [eye_screen[0],eye_screen[1],self.eyeWidth,self.eyeHeight]
```

Q2.f) La méthode mouth

La méthode mouth permet l'affichage de la bouche de l'émoticon, elle prend en paramètre un paramètre x qui dessine une bouche en fonction de « l'humeur de l'émoticon ».

Elle se dessine en fonction du modèle suivant :





Pour la gestion des différentes humeurs de l'émoticon il est important de prendre en comptes plusieurs cas :

- Le cas ou le paramètre est entre -0.15 et 0.15, avec une humeur neutre
- Le deuxième cas ou le paramètre est entre 0.15 et 1, avec un émoticon avec un sourire
- Le dernier cas ou le paramètre peut être entre -1 et -0.15, avec un émoticon avec un sourire triste

Dans le même cas que les fonctions précédentes, l'utilisation de la fonction headToArea() se fait avant la création des arcs ou de la ligne.

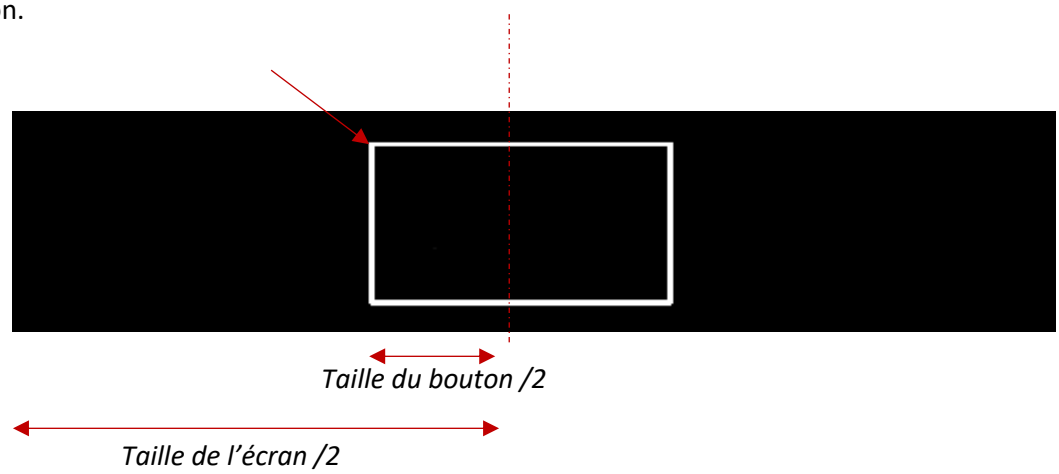
Le résultat, en fonction du paramètre passer dans la fonction est le suivant :

Avec x entre 0.15 et 1 | Avec x entre -0.15 et 0.15 | Avec x entre -1 et -0.15

Q3.a) La méthode getPosition

La méthode getPosition permet de récupérer la position du coin gauche d'un boutton, pour cela il suffit dans un premier temps de connaitre sa position lorsqu'il est unique.

Nous devons alors calculer la moitié de la taille de l'écran ainsi qu'enlever la moitié de la largeur du bouton.

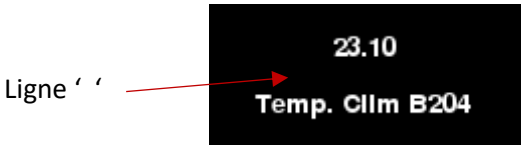


Q3.b) La méthode drawLines()

La méthode drawLines() permet l'affichage de l'ensemble des lignes sur les différents boutons. La mise en place ce fait avec un passage de tableau en parametre (*Exemple : ['line1','line2,'line3,'line4']*), avec un attribut ' ' pour simuler des espaces (lignes vide).

Dans un souci de place et de mise en forme le nombre de ligne est limité à 4. Si le nombre est supérieur à 4, on troncature le tableau.

Exemple pour la fonction drawLines() avec en paramètre : **['22.30',' ','Temp. Clim B204',' ']**



Q4.c) La méthode draw()

La méthode draw permet l'affichage de l'ensemble des informations (appel de la méthode drawLines()) ainsi que l'affichage de la bordure autour du bouton. La fonction draw gère également la position du boutton sur la fenêtre d'affichage.

Partie 4 : Les capteurs

L'objectif de cette partie est la gestion des capteurs, les capteurs interagissent avec les émoticons pour fixer une « humeur » en fonction des températures.

Ces températures intéragissent également avec des seuils.

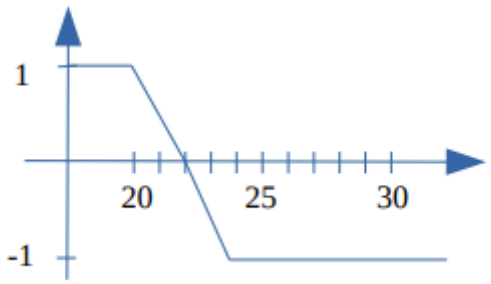
Q4.c) La methode getTransformedValue

L'énoncé nous donnes les capteurs suivants :

Rôle	Nom pour le paramètre *capteur* de l'url	Seuils
Température dans la salle B204 près du climatiseur	epua_b204_clim	[20, 22, 23]
Température dans la salle B204 au niveau de la coursive	epua_b204_coursive	[20, 22, 23]
Température dans la salle B204 au centre	epua_b204_centre	[20, 22, 23]
Température sur le toit de l'école	epua_toiture	[30, 35, 40]
Puissance délivrée par l'onduleur en W	epua_onduleur1_watts	[10000, 12000, 15000]

En fonction des différents capteurs une transformation dite affine est possible, c'est-à-dire qu'on doit jouer avec les seuils pour retourner une valeur dans l'intervalle **[-1,1].**

Nous pouvons prendre en considération l'exemple suivant avec des seuils **[20,25,30]**.



Pour le traitement des différents seuils en fonction des différents capteurs nous pouvons envisager la fonction suivante :

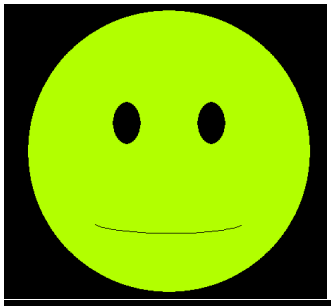
```
def getTransformedValue(self):
      read_value = float(self.read())

      #recupération de l'attribut thresholds
      array_seuils = self.thresholds

      if read_value >= array_seuils[2]:
            return 1
      elif read_value <= array_seuils[0]:
            return -1

      if read_value < array_seuils[1]:
            droite_coeff = 1/(array_seuils[1]-array_seuils[0])
            droite_ordonnee = -droite_coeff*array_seuils[1]
            return read_value+droite_ordonnee*droite_coeff

      droite_coeff = 1/(array_seuils[2]-array_seuils[1])
      droite_ordonnee = -droite_coeff*array_seuils[1]
      return read_value+droite_ordonnee*droite_coeff
```

Dans cette fonction le tableau 'array_seuils' représente les différents seuils du capteur. La fonction retourne donc une valeur entre -1 et 1 qui correspond à la couleur ainsi que l'humeur du personnage.

Exemple de la fonction :

Pour un capteur avec les seuils suivants :
[20, 22, 23]

Pour une température de **22.30**, nous aurons après un passage dans la méthode une valeur de **0.301**, donc le personnage suivant :



Pour un capteur avec les seuils suivants :
[10000, 12000 15000]

Pour une mesure de **56396**, nous aurons après un passage dans la méthode une valeur de -1, car la valeur de mesure est en dehors des seuils prévus , donc le personnage suivant :

Q5.i) La méthode positionToSensorId

La méthode permet de savoir de quel capteur on parle, c'est-à-dire connaitre l'ID du capteur. On vérifie alors si le paramètre position est dans la position dans la zone du bouton.

Si ce n'est pas le cas, la fonction retourne None.

Q5.j) La méthode checkIfSensorChanged

La méthode suivante est finalement un setter, qui vérifie la sélection du sensor pour modifier l'attribut selectedSensor.

```
def checkIfSensorChanged(self, eventPosition):
      if self.positionToSensorId(eventPosition) != None:
            self.selectedSensor = self.positionToSensorId(eventPosition)
```

Q5.k) La méthode isSelected

```
def isSelected(self):
      return self == self.generalConfiguration.getselectedSensor()
```

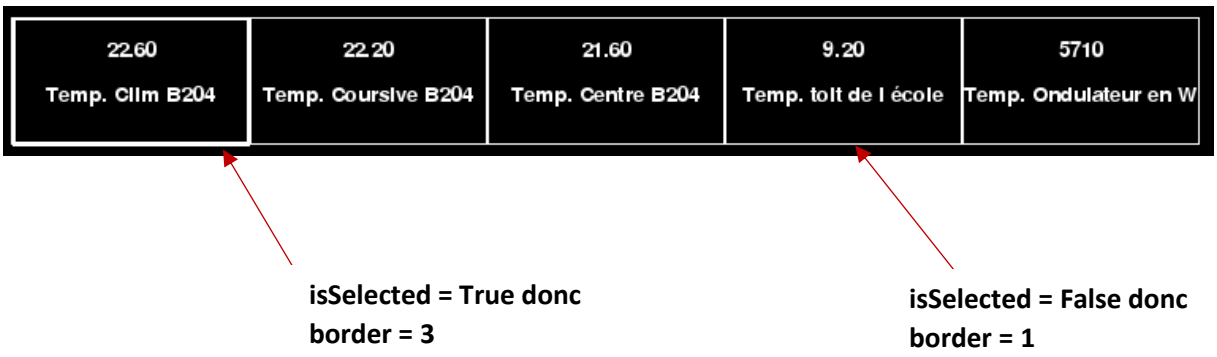
La méthode permet de savoir si le sensor actuel est le sensor qui est selectionner.

L'utilisation de cette méthode permet dans l'affichage du boutton, d'augmenter la taille du bord s'il est selectionné.

Dans la fonction draw du bouton nous pouvons alors l'utiliser de la façon suivante :

```
#Initialisation de la taille du bord à 1px de base
border_size = 1

# Verification si le sensor est selectionner, si oui augmentation de la
taille de la bordure à 3px
if(self.sensor.isSelected()):
   border_size = 3

# Affichage du rectangle en fonction des différents paramètrse de position
et de la taille de la bordure

pygame.draw.rect(self.generalConfiguration.getScreen(), [255,255,255],
[button_get_position[0],button_get_position[1],button_width,button_height],
border_size)
```

Le résultat sera alors le suivant :



isSelected = True donc border = 3

isSelected = False donc border = 1

Résultats finaux

Nous pouvons maintenant tester l'ensemble de nos fonctions avec différents sensors.

En prenant la liste des sensors et des différents seuils, nous pouvons alors avoir la liste des sensors :

```
generalConfiguration.addSensor(
        Sensor(
            'https://www.polytech.univ-
smb.fr/apps/myreader/capteur.php?capteur=epua_b204_clim',
            'Temp. Clim B204',
            [20, 22, 23]
        )
    )

    generalConfiguration.addSensor(
        Sensor(
            'https://www.polytech.univ-
smb.fr/apps/myreader/capteur.php?capteur=epua_b204_coursive',
            'Temp. Coursive B204',
            [20, 22, 23]
        )
    )

    generalConfiguration.addSensor(
        Sensor(
            'https://www.polytech.univ-
smb.fr/apps/myreader/capteur.php?capteur=epua_b204_centre',
            'Temp. Centre B204',
            [20, 22, 23]
        )
    )

    generalConfiguration.addSensor(
        Sensor(
            'https://www.polytech.univ-
smb.fr/apps/myreader/capteur.php?capteur=epua_toiture',
            'Temp. toit de l école',
            [30, 35, 40]
        )
    )

    generalConfiguration.addSensor(
        Sensor(
            'https://www.polytech.univ-
smb.fr/apps/myreader/capteur.php?capteur=epua_onduleur1_watts',
            'Temp. Ondulateur en W',
            [10000, 12000, 15000]
        )
    )
```

En fonction des différents capteurs et de leurs seuils nous pouvons donc avoir l'affichage suivant :


22.60
Temp. Clim B204
22.10
Temp. Coursive B204
22.60
Temp. Centre B204
8.60
Temp. toit de l école
5549
Temp. Ondulateur en W