Mini projet 1 p125

Cahier des charges

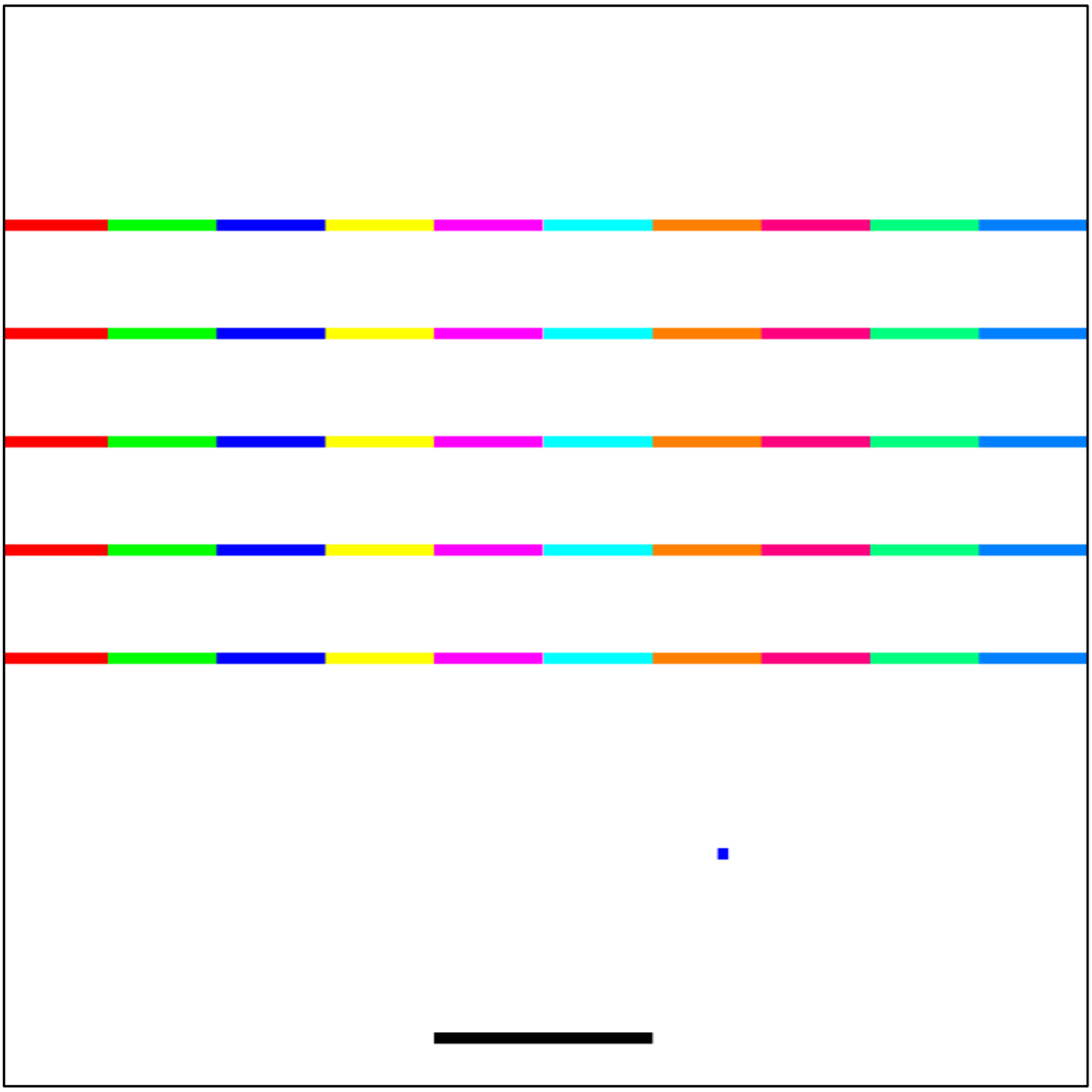
Voir l'aperçu dans cette vidéo astrovirtuel.fr/nsi/p125-1.mp4

Description du panneau de jeu

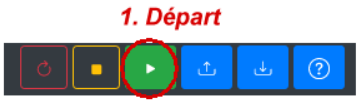
Le panneau est carré, de dimension 100 x 100 pixels. Le pixel de coordonnées $(0, 0)$ est en haut à gauche. Le pixel de coordonnées $(99, 99)$ est en bas à droite.

Il y a une raquette vers le bas de largeur 20 pixels qui est déplaçable lorsque l'utilisateur appuie sur les flèches de direction gauche et droite.

Il y a 5 rangées de briques colorées qui sont détruites lorsque la balle rentre en collision avec elles.

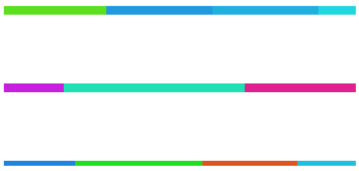


- Départ du jeu :



L'utilisateur doit

1. Cliquer sur le bouton vert en haut, au-dessus du panneau de jeu. La balle commence aussitôt un trajet vers le haut.
2. Valider le déplacement de la raquette en cliquant dans le panneau de jeu.
3. La raquette peut être alors déplacée avec les flèches gauche et droite du clavier.



- Déroulement du jeu

Lors d'une collision de la balle avec un brique, la balle rebondit en détruisant la brique.

Lors d'une collision de la balle avec la raquette, la balle rebondit.

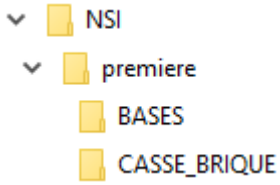
- Fin du jeu

Si la balle sort vers le bas lors de son prochain déplacement alors la partie est perdue et le jeu s'arrête. La raquette est désactivée.

Mise au point de l'Interface Homme-Machine

Votre travail consiste à **compléter successivement huit des neuf fonctions IHM** dans l'ordre indiqué dans le commentaire en haut du programme.

Créez un dossier CASSE-BRIQUE dans votre dossier premiere



- Récupérez le code en ligne comme indiqué sur la page 125 de votre cahier de première NSI (mini-projet 1)
- Enregistrez le dans votre dossier CASSE_BRIQUE sous le nom **casse_brique.py.**
- Ensuite à chaque modification importante, téléchargez le script avec le bouton "flèche vers le bas" et enregistrez-le sous :
  casse_brique1.py
  casse_brique2.py
  etc.
  En cas de besoin vous pourrez ainsi repartir d'une version précédente.

- Commencez par insérer deux lignes de commentaires :

```
#  Script CASSE_BRIQUE
#  Auteur: <ICI VOTRE Prénom Nom>

from game import set_pixel, get_pixel, color
from random import randint
```

Voici l'énoncé : Compléter les fonctions

```
from game import set_pixel, get_pixel, color
from random import randint

"""
L'origine du repère est en haut à gauche, de coordonnées (0, 0).
La zone d'affichage fait 100 * 100 pixels. Le pixel en bas à droite a pour coordonnées (99, 99).

La fonction set_pixel(x, y, couleur) permet de colorier le pixel de coordonnées (x, y). La couleur est définie selon le système rouge, vert, bleu (rgb = red, green, blue) qui nécessite d'indiquer une valeur entière
comprise entre 0 et 255 pour chacun des trois paramètres. Si on indique color(255,0,0) on obtient un rouge pur, pour un gris clair, ce sera par exemple color(150,150,150) et un pourpre color(158,14,64). Le noir est color(0,0,0) et le blanc, à l'opposé, est color(255,255,255).
Pour effacer un pixel, on lui attribue une couleur vide : ""

La fonction get_pixel(x, y) permet de récupérer la couleur du pixel en (x, y). Attention,
ce n'est pas ce qui a été attribué avec set_pixel. La valeur retournée est
destinée à être comparée à "" pour déterminer si le pixel n'est pas colorié, ou à une autre
valeur retournée par get_pixel pour déterminer si deux pixels sont de la même couleur.

Pour démarrer écrire en premier les fonctions "touche_appuyee" et "trace_raquette", ou bien la fonction "deplace_balle". Une fois ces trois fonctions écrites, vous pouvez continuer avec "teste_collision_raquette" et "perdu". Ensuite, il ne reste plus qu'à rajouter la création des briques dans la fonction "initialise", et leur destruction dans "teste_collision_briques" et "efface_brique".

Les variables globales ci-dessous permettent de stocker l'état du jeu.
Il ne faut pas modifier leurs noms.
Pour modifier leur valeur à l'intérieur d'une fonction, il faut utiliser le mot clé "global".
Les valeurs définies ici correspondent au début du jeu.
"""

x_balle = 50 # Coordonnée X de la balle
y_balle = 94 # Coordonnée Y de la balle
d_x_balle = 1 # Déplacement en X de la balle (-1 ou 1)
d_y_balle = -1 # Déplacement en Y de la balle (-1 ou 1)

x_raquette = 40 # Coordonnée X du côté gauche de la raquette
y_raquette = 95 # Position en Y de la raquette
l_raquette = 20 # Largeur de la raquette
```

1. Écrire la fonction touche_appuyee(code_touche)

```
def touche_appuyee(code_touche):
    global x_raquette
    """
    Cette fonction est appelée lorsqu'une touche du clavier est pressée.
    "code_touche" est un entier qui vaut 39 pour la touche "flèche droite"
    et 37 pour la touche "flèche gauche"

    À faire : déplacer la raquette d'un pixel à gauche ou à droite en fonction
    des actions du joueur.
    """
```

6. Écrire la fonction initialise()

```
def initialise():
    """
    Cette fonction est appelée une fois au démarrage de la partie

    À faire : Créer des briques en coloriant des pixels entre les lignes 20 et 60.
    On considère qu'une brique fait toujours un pixel de haut, et s'étend horizontalement
    sur tous les pixels de même couleur. Les briques doivent faire 10 pixels de large.
    Il n'est pas nécessaire de remplir toutes les lignes.
    """
```

```
def boucle():
    """
    Cette fonction est appelée toutes les 0,1 seconde une fois la partie initialisée.
    Elle met à jour l'état du jeu en fonction de l'état précédent et des commandes du joueur.
    Il n'est pas nécessaire de la modifier.
    """
    if not perdu():
        trace_raquette()
        teste_collision_raquette()
        teste_collision_briques()
        deplace_balle()
```

5. Écrire la fonction perdu()

```
def perdu():
    """
    Cette fonction détermine si la partie est perdue. La partie est perdue si la
    balle sort en bas de l'écran lors de son prochain déplacement.

    À faire : retourner "True" si la partie est perdue, "False" sinon.
    """
    return False
```

2. Écrire la fonction trace_raquette()

```
def trace_raquette():
    """
    Cette fonction affiche la raquette à sa position courante. Attention de ne pas
    oublier de l'effacer de sa position précédente !

    À faire : réaliser le comportement décrit ci-dessus
    """
```

4. Écrire la fonction teste_collision_raquette()

```
def teste_collision_raquette():
    global d_y_balle
    """
    Cette fonction détermine si la balle va toucher la raquette lors de son prochain dépla
cement.
    Si c'est le cas, elle inverse le sens de déplacement de la balle (rebond)

    À faire : réaliser le comportement décrit ci-dessus
    """
```

Indication

Pour tester si le pixel qui est sous la raquette est vide, on se sert de la fonction `get_pixel()` ainsi :

```
    pixel_sous_balle = get_pixel(x_balle, y_balle+1)
    pixel_vide = get_pixel(0, 0)
    test_si_pixel_sous_balle_vide = pixel_sous_balle == pixel_vide
```

7. Écrire la fonction teste_collision_briques()

```
def teste_collision_briques():
    global d_x_balle, d_y_balle
    """
    Cette fonction doit déterminer si la balle touche une brique lors de son prochain
    déplacement.
    Si c'est le cas, il faut effacer la brique concernée (en faisant appel à la fonction
    efface_brique(x, y)), et inverser le sens de déplacement de la balle (rebond).
    Pour rappel, voir la définition d'une brique dans la documentation de la
    fonction "initialise".

    À faire : réaliser le comportement décrit ci-dessus
    """
```

8. Écrire la fonction efface_brique()

```
def efface_brique(j, i):
    """
    Cette fonction efface la brique correspondant à liste_briques[i][j]. Pour rappel,
    voir la définition d'une brique dans la documentation de la fonction "initialise".

    À faire : réaliser le comportement décrit ci-dessus
    """
```

3. Écrire la fonction deplace_balle()

```
def deplace_balle():
    global x_balle, y_balle, d_x_balle, d_y_balle
    """
    Cette fonction doit gérer le déplacement de la balle :
     - effacer l'ancienne position
     - déterminer s'il y a rebond sur un des cotés
     - déplacer la balle
     - l'afficher à sa nouvelle position

     À faire : réaliser le comportement décrit ci-dessus
    """
```