Fiche 3-1 : exercices graphiques

Dans cette fiche, vous allez réaliser des programmes graphiques en vous aidant sur la `libgrapythk` (voir mode d'emploi).

Comment utiliser la `libgrapythk` ?

☐ 0. Pour utiliser la `libgrapythk` dans vos programmes :

(a) Assurez-vous d'avoir, dans le même répertoire que votre programe, le fichier `ligrapythk.py` qui vous est fourni sur Moodle.

(b) Dans votre programme, placez en tête de fichier la commande `import libgrapythk`

`libgrapythk` sur votre machine. Pour que la `libgrapythk` fonctionne sur votre machine, vous devez disposer de *python3* et avoir installé les modules `python3-tk` et `PIL`.

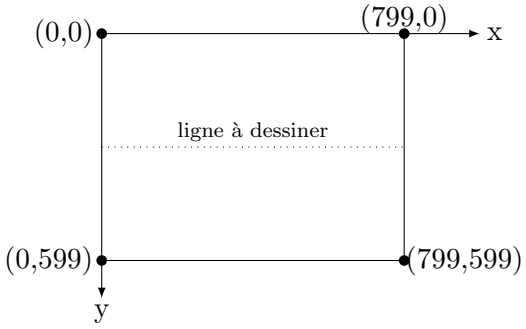
☐ 1. **Premier exemple.** Lisez le code source qui vous est fourni avec la `libgrapythk` dans `main.py`, puis exécutez ce script. Le programme se déroule en même temps dans une fenêtre graphique et dans le terminal. Essayez de bien comprendre le lien entre ce qui se déroule et le contenu de `main.py`.

☐ 2. Lisez attentivement la première page de la notice d'utilisation de la `libgrapythk` à la fin de ce document.
C'est votre manuel de référence pour la librairie graphique, vous devez toujours l'avoir sous la main.

C'est parti...

Quelques petits apéritifs afin de trouver ses marques

Se repérer dans la fenêtre graphique. Dans notre exemple précédent, la fenêtre possède une taille de 800 pixels horizontalement et de 600 pixels verticalement. Le schéma suivant vous indique les coordonnées des quatre coins de la fenêtre pour vous aider à vous repérer, ainsi que la position de votre première oeuvre graphique : la ligne que vous devrez dessiner dans le prochain exercice...



☐ 3. **Ligne horizontale** Ouvrez le programme `exo1.py`. Complétez ce programme (qui pour l'instant ne fait qu'ouvrir une fenêtre, attendre un clic et la refermer) de sorte qu'il dessine une ligne horizontale verte divisant l'écran en 2. Pour vous aider, lisez la notice d'utilisation : vous devez utiliser la fonction `dessinerLigne`.

☐ 4. **Ligne verticale.** Toujours dans `exo1.py`, ajoutez une ligne verticale bleue coupant l'écran en deux.

☐ 5. **Diagonales.** Tant que vous y êtes, tracez les deux diagonales de l'écran.

☐ 6. **Carrément** Tracez un carré (vide) de côté 100 avec quatre lignes, **centré sur l'intersection des diagonales**.

Les couleurs. Un bon paquet de couleurs sont prédéfinies et il suffit d'écrire leur nom comme `blue, green` ou `magenta` pour les utiliser. Voir la liste des couleurs à la fin de la notice.

☐ 7. **Un pixel blanc.** Dans un nouveau fichier `exo_2`, coloriez en blanc le pixel de coordonnées (150,120) à l'aide de la fonction `changerPixel`.

☐ 8. **Pointillés.** On souhaite maintenant allumer en blanc tous les pixels d'ordonnée 50 et d'abscisse paire comprise entre 100 et 200, c'est à dire les pixels de coordonnées `(100,50), (102,50), (104,50), (106,50)`...Utilisez une boucle ! (vous êtes balaise en boucles maintenant)

Des boucles

☐ 9. **10 sur 10.** On passe à l'`exo_3`. Dessinez sur l'écran un quadrillage composé de lignes horizontales et de lignes verticales séparées de 10 pixels. Couleurs aux choix...

☐ 10. **C'est vraiment cyan.** `exo_4` : colorez un par un en cyan les pixels de tout l'écran avec `changerPixel`. Vous ne voyez rien ? Vous devez probablement appeler `actualiser` (voir la notice).

Allez y mollo avec `actualiser` : cette fonction implique un rafraîchissement de tout l'écran, qui est assez coûteux en temps machine. Il est beaucoup plus rapide de n'actualiser qu'une fois toutes les modifications faites.
Si votre exercice précédent est trop lent, c'est très probablement que vous avez placé `actualiser` dans votre double boucle. Essayez de le sortir des boucles et appréciez la différence.

☐ 11. **Tricolore.** `exo_5` : Colorez un par un les pixels de tout l'écran en alternant entre trois couleurs de votre choix (utiliser l'opérateur modulo %). Par exemple, les pixels seront vert, jaune, rouge, vert, jaune, rouge, etc. Et quand on revient à la ligne on utilise la couleur qui devrait suivre le dernier point de la ligne précédente. Remarque : 800 n'étant pas divisible par 3, ceci tracera des lignes de couleur qui seront en diagonale et non verticales.

☐ 12. **Tripentacolore.** `exo_6` : idem mais en alternant entre trois couleurs de votre choix tous les cinq pixels(cinq pixels de la première couleur, cinq de la deuxième, etc).

Un peu d'interaction avec l'utilisateur

☐ 13. **Carré au clic.** `exo_7`Votre objectif : attendre un clic de l'utilisateur et tracer avec `dessinerRectangle` un carré de côté 50 centré sur le point cliqué.
Besoin d'aide ? : allez voir le mode d'emploi de `attendreClic` pour découvrir comment récupérer les coordonnées cliquées.

☐ 14. **5 carrés au clic.** Modifiez la question précédente de sorte que l'on répète 5 fois cette action (donc cinq carrés au final). Avec une boucle, et non pas en répétant cinq fois le même code !

☐ 15. **Effacer les précédents.** Modifiez la question précédente de sorte qu'à chaque fois on efface le carré précédent.

Effacer un objet, c'est simplement le supprimer. Voir la notice...

☐ 16. **Rectangle défini à la souris.** Complétez votre programme afin d'attendre ensuite deux clics de l'utilisateur pour dessiner un rectangle dont les coins sont les deux points cliqués (haut-gauche puis bas-droite).

☐ 17. **Deux conditions.** Dans l'`exo_8` : tracer une ligne blanche qui sépare verticalement l'écran en deux, puis comme précédemment afficher un carré là où clique l'utilisateur. Contrainte supplémentaire : si on clique à gauche de la ligne, le carré sera vert, sinon à droite il sera bleu.

☐ 18. **Trois conditions.** `exo_9` : séparer l'écran en deux comme précédemment. Attendre trois clics ; si les trois clics sont du même côté alors afficher un carré bleu autour du dernier clic, sinon afficher un carré rouge.

Le jeu du monstre

On passe aux choses sérieuses : nous allons concevoir un petit jeu graphique à la souris.

☐ 19. **Le terrain de jeu et le joueur.** `exo_10` : Dessinez un quadrillage de l'écran par des lignes séparées de 20 pixels (reprenez et modifiez le code de l'`exo_3`). Faites en sorte que lorsque l'utilisateur clique dans un carré, l'intérieur de celui-ci devienne rouge. **Attention**, le rouge ne doit pas déborder sur le quadrillage ni laisser d'espace vide.
De l'aide ? Quand on clique dans un carré, vous récupérez une position qui n'est généralement pas le coin haut-gauche dont vous avez besoin pour dessiner ce carré. Il vous faut alors effectuer un calcul afin d'obtenir cette position. Il se pourrait que le modulo vous soit utile...

☐ 20. **Déplacement du joueur.** Modifiez ce programme de sorte que l'utilisateur puisse « sélectionner »un carré et changer d'avis : lorsqu'il clique dans un carré, celui-ci devient rouge, mais s'il clique dans un autre carré, le carré précédent disparait et le nouveau devient rouge, ceci 10 fois de suite.

☐ 21. **Un terrain plus joli.** `exo_11` : Même exercice que le précédent, mais plutôt qu'un quadrillage, l'écran est un damier constitué de carrés verts et bleus de côté 20. Chaque carré doit reprendre sa couleur de départ quand on en sélectionne un autre.

☐ 22. **Sélection du voisinage.** `exo_12` : Comme l'exercice précédent, mais maintenant l'utilisateur ne peut choisir qu'un des carrés voisins (diagonale possible). Si l'utilisateur clique dans un carré non voisin alors rien ne se passe.

☐ 23. **Le très horrible monstre jaune.** Ajoutez un carré de couleur jaune qui apparaît dès le lancement du programme sur une position aléatoire du damier. Ensuite, quand votre joueur rouge se déplace, il lui est impossible d'aller sur la case jaune.

☐ 24. **Le jeu !** `exo_12` : Tout est en place, ne reste qu'à lancer la mécanique : à chaque fois que le joueur rouge se déplace dans son voisinage, le monstre se déplace également automatiquement d'une case *vers le joueur* (au moins de façon à se rapprocher). Si après son mouvement le monstre est sur la case du joueur, le programme s'arrête et le joueur a perdu.

☐ 25. **Make it harder.** (A faire seulement si vous êtes en avance) Et si maintenant il y avait deux, ou trois monstres ? Ou des murs ?

Validation : faites nous voir ça

Déplacement au clavier

☐ 26. `exo_13` : Lire dans la notice comment utiliser la fonction `recupererTouche`. Dessinez un carré de côté 50, que l'on peut déplacer avec les flèches du clavier. *Indication : Dans une boucle, attendre qu'une touche soit appuyée et déplacez le carré de quelques pixels suivant la touche appuyée.*

Attention, le carré ne doit pas pouvoir "sortir" de l'écran et s'il rencontre un bord d'écran, il doit être bloqué dans cette direction.

☐ 27. Améliorez le code précédent en faisant en sorte que le programme se termine au cas où on appuie sur la touche espace.

Du mouvement : les carrés rebondissants

☐ 28. **Un carré dans le mur.** `exo_14` : dessinez un carré de côté 50 là où clique l'utilisateur ; le carré se déplace alors en suivant le vecteur (5,5) (c'est-à-dire que x augmente de 5 et y augmente de 5) jusqu'à atteindre le bord. Veillez à bien terminer votre boucle au moment exact où le carré atteint le bord !
Remarque : ici, vous pouvez utiliser, en début de programme, des variaeasble globales pour les coordonnées du vecteur, afin de pouvoir changer facilement le vecteur vitesse ultérieurement.
Attention ! Est-ce que votre programme fonctionne quelle que soit la position de départ du carré ?

Le dilemme du programmeur : « *monteriez-vous dans un avion piloté par un programme que vous avez écrit ?* »Vous devez systématiquement chercher à imaginer tous les cas possibles et à les résoudre.

☐ 29. **Rebond.** `exo_15` : même exercice, mais cette fois-ci le carré rebondit sur le bord. Le vecteur de déplacement va donc changer : suivant les cas, il s'agira d'un des quatre vecteurs de la forme $(\pm 5, \pm 5)$. A vous de trouver lequel pour que cela ait l'air d'un rebond réaliste.

☐ 30. **Rebond++.** Si vous essayez avec le vecteur vitesse $(2, 5)$, les rebonds auront peut-être l'air étranges... Reprendre dans `exo_16`.c l'exercice précédent mais faites en sorte qu'il fonctionne pour n'importe quel vecteur de déplacement.

☐ 31. **Deux carrés qui s'ignorent.** `exo_17` : Idem avec deux clics qui donnent donc deux carrés ; dans un premier temps on peut supposer que les carrés se croisent sans se toucher.

☐ 32. **Choc de carrés.** Améliorez le programme précédent de sorte que les carrés rebondissent l'un sur l'autre quand ils se touchent. Ce n'est pas très réaliste, mais on peut supposer que les carrés repartent en sens inverse quand ils se touchent. Ensuite si vous êtes vraiment en forme vous pouvez essayer de simuler une véritable collision (plus difficile mathématiquement).