Documentation MLIB

Présentation :

Mlib est une petite bibliothèque, crée par Mattéo Menou, 1°5, dans le cadre enseignement de spécialité NSI au lycée. Elle doit regrouper de multiples choses permettant de faciliter l'accès d'un élève « moyen » à des programmes plus complexes. Il contient pour l'instant une plateforme graphique, nommé « MGUI ».

Note 1 : La version actuelle de Mlib est une version adapté au projet de cryptographie de NSI, rendu le 31/11/2022. Il est déconseillé d'utiliser celle là pour quelconque autre usage, à moins d'avoir très bien compris l'utilisation de cette dernière.

Note 2 : Les versions suivantes de Mlib se trouveront sur mon github : Mattool81974 (github.com) ou sur mon site web : Matt_o (mattool81974.github.io).

Pour fonctionner, elle nécessite plusieurs bibliothèques :

-pygame
-pygame.init()
-sys
-math
-os
-time (que la fonction time_ns())

Remarques : les méthodes privées sont fournis pour mieux comprendre la lib mais ne sont pas destiné à l'utilisation.

MGUI :

MGUI contient plusieurs choses permettant de construire une fenêtre graphique des plus qualitatives.

MWidget :

Mwidget est la classe principale de MGUI, permettant de construire tous les éléments graphiques présent dans la bibliothèques. Toutes les autres classes hérité de lui.
Les Mwidgets utilisent un système de hiérarchie avec des widgets fils et parents, pour permettre de mieux organiser la fenêtre graphique. Cependant, si un Mwidget est invisible, c'est Mwidgets enfants aussi. De plus, la position d'un Mwidget est relative à son parent.

Méthodes publiques :

__init__(self, taille, position, ...) :

Constructeur de Mwidget, permettant d'initialiser un widget normale. Elle prend une multitude de paramètres permettant d'initialiser le widget en un seul coup.

Paramètres : taille pour définir la taille du widget, position pour définir sa position par rapport à son parent, parent pour définir son parent défini de base à None, arrierePlanCouleur pour définir la couleur de l'arrière plan de base blanche, curseurSurvol pour définir la texture du curseur quand il survolera le Mwidget, de base à SYSTEM_CURSOR_ARROW (la flèche), type pour définir le type de widget.

get_parent() :

Retourne le parent du Mwidget en objet Mwidget.

get_position() :

Retourne la position du Mwidget en tuple de 2 de long.

get_rect() :

Retourne le rect du Mwidget, c'est à dire la position + la taille en tuple de 4 de long.

get_survol() :

Retourne si le Mwdiget est survolé ou non en booléan.

get_taille() :

Retourne la taille du Mwidget, en tuple de 2 de long.

get_visible() :

Retourne si le Mwidget est visible, en booléen.

set_parent(parent) :

Supprime le parent actuel est met parent à la place.

Paramètres : Le nouveau parent de type Mwidget du Mwidget.

set_taille(taille) :

Change la taille actuel en taille.

Paramètres : La nouvelle taille de type tuple de 2 de long du Mwidget.

get_visible() :

Change la visibilité du Mwidget.

Paramètres : La visibilité du Mwidget de type booléen.

Méthodes privées :

__enleverParent(enfant) :

Supprime enfant de la liste d'enfants.

Paramètres : L'enfant à enlever.

__nouveauEnfant(enfant) :

Paramètres : L’enfant à ajouter.

_render() :

Effectue le rendu du Mwdiget et le retourne sous forme de surface.

Cette fonction va commencer par appliqué à la surface du Mwidget la couleur d’arrière plan, puis le curseur de survol, puis va appeler _renderBeforeHierarchy(surface), puis va afficher le retour du _render() des enfants sur la surface à retourner (si ils sont visibles), puis va appeler _renderAfterHierarchy(surface), puis va retourner la surface. Elle va ensuite ré-initialiser le focus pour ne pas que des comportements inattendus arrives.

Note : cette fonction va bientôt être modifié pour l’optimisation, en transformant la surface de base, qui va être modifié, recréer et appelé de multiple fois, par un texte, qui sera traité par une classe pour gérer l’application.

_renderBeforeHierarchy(surface) :

Fonction faites pour être ré-implémenter dans des classes hérité, pour afficher quelque chose sur la surface avant la gestion de la hiérarchie.

Paramètres : La surface sur la quel appliqué le renduu.

_renderAfterHierarchy(surface) :

Fonction faites pour être ré-implémenter dans des classes hérité, pour afficher quelque chose sur la surface après la gestion de la hiérarchie.

Paramètres : La surface sur la quel appliqué le rendu.

Attributs :

self.arrierePlanCouleur :

Contient l’arrière plan de la couleur.

self.curseurSurvol :

Contient le curseur de la souris en cas de survol.

self.enfant :

Liste contenant l’entièreté des enfants du Mwidget.

self.fenetrePrincipale :

Contient la fenêtre principale sur lequel le Mwidget est posé.

L’affectation de cette variable à lieu dans __init_(), en regardant tous les parents du parent du parent… jusqu’à arriver à la fenêtre principale. Si rien est trouvé, la fenêtre principale est défini comme le Mwidget en lui même.

self.focus :

Variable de type booléen indiquant si le widget est focus ou non. Il est OBLIGATOIRE de la modifié dans _renderAfterHierarchy(surface) pour éviter tout comportements inattendus.

self.globalPosition :

Position par rapport à la fenêtre principale, en tuple de 2 de long.

L'affectation de cette variable à lieu en même temps que celle de fenetrePrincipale, en additionnant les positions des parents.

self.parent :

Contient le parent du Mwidget.

self.position :

Contient la position du widget, en tuple de 2 de long.

self.taille :

Contient la taille du widget, en tuple de 2 de long.

self.type :

Contient le type du widget de base à Widget.

MFenetre :

MFenetre permet de définir une fenêtre graphique a afficher à l'écran. Dans les versions actuelles, une seul fenêtre graphique ne peut être créer à la fois et la fenêtre créer représente l'application en elle même.

Méthodes publiques :

__init__(fenetre, titre…) :

Créer un objet de type MFenetre. Il hérite de MWidget.

Paramètres : fenetre pour définir la fenêtre sur laquelle il faut appliquer les modifications, titre pour le titre de l'application (le texte écrit en haut de la fenêtre graphique) de base défini à « Fenêtre MGUI », arrierePlanImage pour avoir le chemin d'accès vers l'image d'arrière plan de base défini à vide, arrierePlanImageAlignement pour savoir comment aligner l'image d'arrière plan sur la fenêtre de base à « GH », arrierePlanImageParSeconde pour savoir à quelle vitesse faire défiler les images si il s'agit d'une image animé (gif), tout les paramètres du constructeur de MWidget.

La taille est défini selon la taille de fenetre et la position à (0, 0). Le parent est à None et le type est défini à « Fenetre ».

Pour utiliser un gif, il faut couper un gif en toutes les images qui le constituent, et le mettre dans un répertoire. Il faut ensuite mettre dans arrierePlanImage « cheminDuRépertoire+.gif ». Il est d'autant

plus conseillé, pour les ordinateurs peu puissants, d'adapter la taille des images pour qu'elle rentre dans la fenêtre sans avoir à les justifier.

frame() :

Effectue une mise à jour de la fenêtre.

La fenêtre étant hérité de MWidget, elle va afficher sur fenetre le retour du _render() de ce Mwidget. Avant ça, elle va voir si un des Mwidget enfant est focus. Elle annule aussi tout les Mwdiget si la souris est clické. Elle va aussi calculer le deltaTime de la fenêtre et les FPS. Elle s'occupe aussi du curseur. Elle stocke aussi la position de la souris dans une variable self.sourisPosition.

Elle utilise les évènements de pygame. Il est donc déconseillé de les utiliser autre part. Elle permet d'initialiser quelques variables de certaines touches du claviers spécifiques (shift, caplock).

get_cursor() :

Retourne le curseur actuelle de la fenêtre.

get_titreFenetre() :

Retourne le titre de la fenêtre.

_nouvelleElement(element) :

Ajoute un élément dans la fenêtre.

set_cursor(curseur) :

Change le curseur actuelle pour curseur.

Paramètres : curseur qui représente le curseur à mettre de type curseur

set_titreFenetre(titre) :

Change le titre de la fenêtre.

Paramètre : titre qui représente le nouveau titre de type string.

Remarque : Il est très conseillé d'utiliser cette fonction plutôt que de modifier directement titreFenetre, car la modification de titreFenetre implique le besoin de redéfinir le titre de la page, ce qui n'est pas garanti.

Méthodes privées :

_renderBeforeHierarchy(surface) :

Fonction ré-implémenter de Mwidget. Elle permet d'afficher les bases de la fenêtre, donc l'image d'arrière plan si il y en a une.

Elle va premièrement affecter à une variable img l'image load de l'image à chargé pour cette frame. Elle est soit une image normale, soit une image d'un gif, alors elle va être choisis selon l'image du gif actuelle, et est changé si le temps depuis la dernière image est trop élevé.

Elle va ensuite s'occuper de placer l'image selon l'alignement de l'image.

Paramètres : La surface sur la quel appliqué le rendu.

Attributs :

self.actuelFrameGif :

Numéro de la frame actuelle du gif, si l'image d'arrière plan est un gif.

self.arrierePlanImage :

Chemin d'accès de type str vers l'image ou le répertoire du gif. Il prend le dessus sur self.arrierePlanCouleur.

self.arrierePlanImageParSeconde :

Nombre d'images par secondes au cas ou l'image d'arrière plan soit un gif.

self.arrierePlanImageParSeconde :

Temps passé en secondes entre la dernière fois que l'image de arrierePlanImage a été changé (si il s'agit d'un gif) en secondes.

self.caplockPressee :

Savoir si la touche caplock (cadenas de majuscule) est pressé en booléen.

self.curseur :

Contient le curseur actuelle de la fenêtre.

self.deltaTime :

Temps en seconde entre le dernier appel de frame() et celui actuel.

self._deltaTime :

Variable tampon pour actualiser self.deltaTime.

self.evenement :

Tout les évènements de la frame actuel.

self.fenetre :

Fenêtre principale du programme défini dans le programme principale sur laquelle appliquer les rendus.

self.fps :

Nombre d'appel de frame() en nombre par seconde.

self.fpsMoyen :

Nombre d'appel de frame() en nombre par seconde en moyenne pour cette fenêtre.

self.fpsNb :

Nombre d'actualisation des fps en nombre pour le calcul de self.fpsMoyen.

self.fpsNbFrame :

Nombre de frame en nombre lors de cette seconde pour l'actualisation de self.fps.

self.positionSouris :

Position de la souris dans un tuple de 2 de long.

self.shiftPressee :

Savoir si la touche shift est pressé en booléen.

self.toutLesElements :

Liste de tout les éléments dans cette fenêtre.

Les attributs hérités :

Les même attributs que Mwidget, avec la même utilisation.

MBordure :

MBordure permet de créer un widget avec (ou sans) une bordure. Elle hérite de MWidget.

Méthodes publiques :

__init__(position, taille, parent...) :

Construit un objet de type Mbordure.

Paramètre : bordureLargeur qui défini la taille de la bordure du widget de base à 2, bordureCouleur qui défini la couleur de la bordure du widget de base à (0, 0, 0), bordureRayon qui défini l'arrondi des angles du widget de base à 0, bordureLargeurDroite qui défini la taille de la bordure droite de base à None, bordureLargeurGauche qui défini la taille de la bordure gauche de base à None, bordureLargeurBas qui défini la taille de la bordure basse de base à None, bordureLargeurHaut qui défini la taille de la bordure haute de base à None, bordureRayonGB qui défini l'arrondi de l'angle en bas à gauche du widget de base à None, bordureRayonGH qui défini l'arrondi de l'angle en haut à gauche du widget de base à None, bordureRayonDB qui défini l'arrondi de l'angle en bas à droite du widget de base à None, bordureRayonDH qui défini l'arrondi de l'angle en haut à droite du widget de base à None, tout les paramètres du constructeur de MWidget.

Les différentes variables du constructeur bordureLargeurDroite, bordureLargeurGauche, bordureLargeurBas, bordureLargeurHaut peuvent être laissé à None. Dans ce cas, leur valeurs dans l'objet seront de bordureLargeur. C'est pareil pour bordureRayon et les variables similaires.

Méthodes privées :

_renderBeforeHierarchy(surface) :

Tracer la bordure sur la surface. Pour des raisons de hiérarchie, la surface est rempli de vide, puis un premier cadre de taille similaire à celle de la bordure et de la couleur bordureCouleur est tracé. Ensuite, une surface de la même couleur que la couleur de l'arrière plan de la bordure est tracé. Sa taille est la même que celle du premier cadre adapté selon la largeur des différents côtés du widget. L'arrondi des angles est aussi pris en compte dans le tracé.

Attributs :

self.bordureCouleur :

Couleur en argb de la bordure.

self.bordureLargeurBas :

Taille de la bordure en nombre entier en bas.

self.bordureLargeurDroite :

Taille de la bordure en nombre entier à droite.

self.bordureLargeurGauche :

Taille de la bordure en nombre entier à gauche.

self.bordureLargeurHaut :

Taille de la bordure en nombre entier en haut.

self.bordureRayonDB :

Niveau d'arrondissement en nombre entier du coin en bas à droite de la bordure.

self.bordureRayonDH :

Niveau d'arrondissement en nombre entier du coin en haut à droite de la bordure.

self.bordureRayonGB :

Niveau d'arrondissement en nombre entier du coin en bas à gauche de la bordure.

self.bordureRayonGH :

Niveau d'arrondissement en nombre entier du coin en haut à gauche de la bordure.

Les attributs hérités :

Les même attributs que Mwidget, avec la même utilisation.

MTexte :

MTexte est une classe permettant de faire le rendu graphique de texte. Elle hérite de MBordure.

Elle met à disposition un curseur qui peut être activé et désactivé pour faciliter l'entrée de texte par l'utilisateur ou pour une quelconque animation.

Méthodes publiques :

__init__(texte, position, taille, parent...) :

Construit un objet de type MTexte.

Paramètres : curseur défini si le curseur est affiché ou non de base à False, curseurLargeur défini la largeur du curseur en nombre entier de base à 2, curseurTempsDAffichage défini le temps que le curseur restera affiché en secondes de base à 0.4, ligneLongueurMax défini la taille maximale d'une ligne en nombre entier de base à -1, ligneMax défini le nombre maximale de ligne en nombre entier de base à 1, longueurMax défini la taille maximale du texte en nombre entier de base à -1, policeTaille représente la taille de la police en nombre entier de base à 12, policeType représente le nom de la police du texte en str de base à Arial, texteAlignement représente l'alignement du texte en str dans l'entrée de base à GH, texteCouleur en argb qui défini la couleur du texte de base à (0, 0, 0), tout les paramètres de MBordure.

Si la longueur de ligne max est égal à -1, alors la longueur de la ligne est celle du widget moins celle des bordures. Si la police est Default, alors la police de base de l'ordinateur est choisis.

get_texte() :

Retourne le texte actuelle du widget.

set_curseurPosition(position) :

Mais la position actuelle du curseur à position.

set_texte(texte) :

Mais le texte actuel à texte.

Méthodes privées :

_renderBeforeHierarchy(surface) :

Génère tous les éléments à afficher sur la surface du texte.

Pour un soucis d'esthétique, le texte qui sera affiché comme résultat n'est pas exactement le même que celui de base. En effet, la fonction va tout d'abord actualiser l'état du curseur (affichage ou non, temps d'affichage…). Ensuite, elle va actualiser la police pour qu'elle correspondent au données voulues. Ensuite, le texte va être découpé pour pouvoir rentrer dans chaque ligne comme

ligneLongueurMax, longueurMax (si différent de -1) et ligneMax le défini. Si une de ces 3 variables sont dépassé, alors le texte en trop est supprimé ou remis à la ligne dans le cas de ligneLongueurMax (si ligneMax le permet). Ensuite, le rendu graphique du texte est crée. La position précise du curseur sur l'écran est d'ailleurs donnée à ce moment précis. Ensuite, les coordonnées et tailles du texte sont définis. Finalement, le texte est mis sur la surface finale.

Attributs :

self.curseur :

Défini en un booléen si le curseur doit être affiché ou pas.

self.curseurLargeur :

Défini en nombre entier la largeur du curseur.

self.curseurPosition :

Défini la position du curseur.

self.curseurTempsDAffichage :

Défini en secondes le temps d'affichage ou de non affichage du curseur.

self.curseurTempsDAffichageAffiche :

Défini en un booléen si le curseur est affiché ou non.

self.curseurTempsDAffichageEcoule :

Défini en secondes le temps entre le dernier changement d'état et maintenant.

self.ligneLongueurMax :

Défini en pixel la longueur max d'une ligne.

self.ligneMax :

Défini en nombre le nombre de ligne max.

self.longueurMax :

Défini en nombre le nombre de caractères max.

self.policeTaille :

Défini en nombre la taille de la police.

self.texte :

Défini en str le texte du texte.

self.textes :

Contient le texte découpé en saut de ligne.

self.alignement :

Défini en str l'alignement du texte par rapport au widget.

self.texteCouleur :

Défini en argb la couleur du texte.

self.texteRect :

Défini en tuple de longueur 4 le rect du dernier éléments de textes lorsque render est appelé.

Les attributs hérités :

Les même attributs que Mbordure, avec la même utilisation.

MBouton :

Bouton au fonctionnement conventionnel.

Méthodes publiques :

__init__(texte, position, taille, parent...) :

Construit un objet de type MBouton.

Paramètres : actionAuSurvol défini une façon simple de modifier le comportement du bouton lors du survolement, tout les paramètres de Mtexte.

get_clicke() :

Retourne si le bouton est clické pendant cette frame.

Méthodes privées :

_renderBeforeHierarchy(surface) :

Actualise l'aspect du bouton. Si il est survolé, alors l'action de actionAuSurvol est effectué. Si il est clické, alors get_clicke() pour renvoyer True lors de cette frame.

Attributs :

self.actionAuSurvol :

Action qui sera effectué au survol en str.

La syntaxe du str doit être très précise : un attribut à modifier et sa valeur prochaine. Pour l'instant, les attributs modifiables sont « policeTaille » et « texte », tout deux de MTexte.

MentreeTexte :

Une version de MTexte modifiable par l’utilisateur.

Méthodes publiques :

__init__(texte, position, taille, parent...) :

Construit un objet de type MEntreeTexte.

Paramètres : caracteresAutorises défini les caractères autorisée entrable par l’utilisateur dans un str, tout les paramètres de Mtexte.

Méthodes privées :

_renderBeforeHierarchy(surface) :

Actualise l’aspect de l’entrée. Avant cela, les évènements d’entrées sont appliquées. Seules les caractères autorisées sont pris en compte, mais si caracteresAuthorises est à all, tout les caractères possibles et imaginables sont autorisées.

Attributs :

*self.*caracteresAutorise :

Défini tous les caractères autorisées.

Nomenclature :

La bibliothèque utilise certaine nomenclature.

Alignement :

Les alignements de texte, image… respectent une nomenclature très précise. Toutes les variables contenant « Align » ou « Alignement » sont concernées par cette nomenclature. Il s’agit d’un texte composé de 2 caractères. Le premier caractère est l’alignement horizontal, et le deuxième l’alignement vertical. L’alignement est défini selon le caractère : G pour la gauche, D pour la droite, C pour le centre (pour horizontal et vertical), J pour justifier (adapter la taille selon le widget, pas supporté par tout les widgets), H pour haut, B pour bas.