Mattéo Menou, Terminal 5

Rapport de projet

Projet

Le projet consiste en un petit jeu. Il s'agit d'un jeu reprenant le principe d'un stand de tir où les cibles sont des balles en mouvement.

Au début, l'utilisateur se trouve sur une page d'accueil avec 2 boutons : « jouer » (pour commencer la partie) et « comment jouer » (pour apprendre les bases du jeu).

Dans « comment jouer », l'utilisateur à accès au bases du jeu ainsi qu'une présentation des armes utilisables.

Dans « jouer », l'utilisateur est amené à une page où il indiquera les configuration de la partie (arme utilisée et nombre de balle). Il a accès à 2 boutons : « jouer » (pour entrer dans la partie) et « retour » (pour retourner à l'accueil.

Dés que l'utilisateur sera dans la partie, il se retrouvera à tenir son arme (par le manche) et devra détruire les balles en pointant son arme sur elle (les munitions iront au niveau de la croix blanche au dessus de l'arme). Chaque arme à un chargeur, avec les données respectives au chargeur en bas à gauche de l'écran. Si le chargeur est vide, il peut le recharger avec la touche R. En haut à droite se trouve un chronomètre qui indique à l'utilisateur le temps qu'il a passé à la partie et en bas à gauche l'inscription « projet NSI ». En bas se trouve une excroissance permettant de tenir son arme dans les parties les plus basses de la zone de jeu. Lorsqu'une balle touche un ballon, il explose. Si la balle est une roquette, tout les ballons dans un rayon de 150 pixels. Lorsque toutes les balles sont détruites, il attend 1 seconde et est téléporté à la dernière page.

La dernière page contient les statistiques de l'utilisateur (balles détruites, tirs, pourcentage de tir réussi et temps passé). Il a accès à 2 boutons : « rejouer » (pour être mis à la page d'accueil) et « quitter » (pour quitter le jeu).

Choix de programmation

Les définitions :

J'ai utilisé pour l'interface graphique ma propre bibliothèque graphique « MLib » version 2.0.1 pour simplifier l'affichage graphique. Elle est importée ligne 2 à 3. On définit la taille de la fenêtre entre la ligne 7 et 10. De la ligne 12 à 13, on défini la fenêtre graphique de MLib et tout ce dont elle a besoin. De la ligne 20 à 150, on définit la classe Balle.

Classe Balle :

Cette classe offre toutes les données nécessaires à l'affichage d'une balle.

Méthodes :

-__init__(rayon, x, y) : constructeur d'une balle de position (x, y) et de rayon « rayon ».
-centre() : retourne le tuple du centre de la balle.
-chargerTexture() : charge les textures de la balle (attribut texture, attribut TEXTURE, attribut textureRouge et attribut TEXTUREROUGE).
-getDX() : retourne l'attribut dx.
-getDY() : retourne l'attribut dy.
-getMasse() : retourne l'attribut masse.
-getRayon() : retourne l'attribut rayon.
-getRouge() : retourne l'attribut rouge.
-getTexture() : retourne l'attribut texture.
-getX() : retourne l'attribut x.
-getY() : retourne l'attribut y.
-getZ() : retourne 100 (la position de la balle sur l'axe Z ne changeant jamais).
-move(x, y) : change l'attribut x à « x » et l'attribut y à « y ».
-redimensionnerTexture() : redimensionner la texture si nécessaire (et appliquer l'effet rouge).
-setDX(dx) : change l'attribut dx par « dx ».
-setDY(dy) : change l'attribut dy par « dy ».
-setRayon(rayon) : change l'attribut rayon par « rayon ».
-setRouge(rouge) : change l'attribut rouge par « rouge ».
-setX(x) : change l'attribut x par « x ».
-setY(y) : change l'attribut y par « y ».
-touche(balle) : retourne si la balle « balle » touche la balle self ou non.
-touchePoint(x, y, rayon = 0) : retourne si le point coordonnée (« x », « y ») de rayon « rayon » touche la balle self ou non.
-vecteurVitesse() : retourne le vecteur vitesse (attribut dx, attribut dy).

Attributs :

-dx : vitesse sur l'axe x de la balle, basiquement à 0.
-dy : vitesse sur l'axe y de la balle, basiquement à 0.
-masse : masse de la balle, basiquement à 1.
-rayon : rayon de la balle.
-rouge : booléen qui indique si la balle doit avoir une texture rougit ou non, basiquement à False.
-texture : texture de la balle à utiliser (possiblement modifiée)
-textureRouge : texture de la balle rouge à coller à texture pour utilisation (possiblement modifiée)
-x : coordonnée x de la balle.
-y : coordonnée y de la balle.
-TEXTURE : texture de la balle de base sans aucune modification
-TEXTUREROUGE : texture de la balle rougit de base sans aucune modification

Ensuite, de la ligne 154 à 226, on définit la clase Arme.

Classe Arme :

Cette classe offre toutes les données nécessaires à l'utilisation d'une arme.

Méthodes :

-__init_(cadenceDeTir, chargeur, semiAutomatique, tempsDeRechargement, texture, type, munitionTexture, munitionVitesse) : constructeur d'une balle qui met les attributs de même nom que les paramètres à la même valeur que les paramètres.
-debuterChargement() : commence le chargement du chargeur de l'arme
-enChargement() : retourne l'attribut « seRecharge »
-finirChargement() : vérifie si le chargement est terminé, si oui retourne 0 sinon retourne le temps restant
-getCadenceDeTir() : retourne l'attribut cadenceDeTir
-getChargeur() : retourne l'attribut chargeur
-getChargeurRestant() : retourne l'attribut chargeurRestant
-getMunitionTexture() : retourne l'attribut munitionTexture
-getMunitionVitesse() : retourne l'attribut munitionVitesse
-getSemiAutomatique() : retourne l'attribut semiAutomatique
-getTempsDeChargement() : retourne l'attribut tempsReChargement
-getTexture() : retourne l'attribut texture
-getType() : retourne l'attribut type
-reset() : remet l'arme à zéro
-tirer() : simule l'effet d'un tir avec l'arme

Attributs :

-cadenceDeTir : retourne la cadence de tir de l'arme.
-chargeur : retourne la capacité du chargeur.
-chargeurRestant : nombre de munition qu'il reste dans le chargeur.
-debutChargement : moment (time_ns()) de début de chargement.
-munitionTexture : texture de la munition.
-munitionVitesse : vitesse de l'arme (pas de réel unité).
-semiAutomatique : booléen qui indique si l'arme est semi automatique ou pas.
-seRecharge : booléen qui indique si l'arme se recharge ou pas.
-tempsDeRechargement : temps de chargement de l'arme.
-texture : texture de l'arme
-type : nom du type de l'arme

Ensuite, de la ligne 228 à 602, on définit la clase Game.

Classe Game :

Cette classe permet de construire le moteur de jeu qui hérite de MImage dans MLib.

Méthodes :

-__init__(typeArme, x, y, parent, widgetType="Mimage") : crée un moteur de jeu qui utilisera l'arme « typeArme », placé sur le MWdiget « parent » à la coordonnée (« x », « y »).
-balleALaPosition(x, y, rayon = 0) : retourne si une balle de coordonnée (« x », « y ») et de rayon « rayon » existe dans le jeu.

-changerArmeChargeurTexte() : change le texte de « armeChargeurTexte » pour les valeurs du chargeur actuelle.
-changerArme(arme) : change l'arme utilisé dans le jeu.
-creerArme() : crée tous les objets armes disponibles dans le jeu.
-creerBalle(nombre) : crée « nombre » balle de position et vitesse différentes.
-donneesFinDeJeu() : retourne les données nécessaires à l'affichage de fin de jeu (dictionnaire avec nombre de balle « nbBalleDetruite », temps « temps », nombre de tir « nbTir », nombre de tir réussi « nbTirReussie »).
-fin() : fini le jeu.
-framePhysique(deltaTime) : effectue une frame physique (simulation des balles et des munitions) en prenant en compte « deltaTime » qui correspont au temps depuis la dernière frame.
-getCalculerCollision() : retourne si les collisions sont calculées ou pas.
-getCroixDeVisee() : retourne l'attribut croixDeVisee.
-rectFenetreDeJeu() : retourne les coordonnées et taille de la fenêtre de jeu.
-setCroixDeVissee(croixDeVisee) : change l'attribut croixDeVisee par « croixDeVisee ».
-tempsDepuisFin() : retourne le temps depuis la fin de jeu (l'appel de fin()).
-tirer() : tire avec l'arme.
-_isGettingKeyPressed(key) : fonction héritée de MWidget appelé lorsque qu'une touche du clavier est pressée (paramètre « key »), utilisée si « key » == R pour recharger l'arme.
-_isGettingMouseDown(button, pos, relativePos) : fonction héritée de MWidget appelé lorsque qu'une touche de la souris est clické (paramètre « button »), utilisé pour tirer avec l'arme.
-_isGettingMouseUp(button, pos, relativePos) : fonction héritée de MWidget appelé lorsque qu'une touche de la souris n'est plus clické (paramètre « button »), utilisé pour arrêter de tirer avec des armes automatiques.
-_mouseMove(button, pos, relativePos) : fonction héritée de MWidget appelé lorsque la souris bouge (avec paramètres « button » les touches clickée, « pos » la nouvelle position de la souris et « relativePos » le mouvement de la souris) pour bouger l'arme.
-_renderBeforeHierarchy(surface) : fonction héritée de MWidget appelé pour dessiner ce qu'il faut dessiner du widget sur la surface, dessinant les balles, puis les explosions de roquette, puis les munitions, puis la croix de visée, puis l'arme.
-_update(deltaTime) : fonction héritée de MWidget appelé à chaque frame de MLib, s'occupant de gérer le chronomètre, les explosions, le rechargement de l'arme et les tirs de fusils automatiques.

Attributs :

-armes : dictionnaire contenant tous les types d'armes (« ar15 », « glock48 » et « lanceRoquette »).
-armeActuel : type de l'arme actuel.
-armeChargeurTexte : MText du texte de chargeur en bas à gauche.
-armePosition : position de l'arme sur la zone de jeu.
-balles : liste qui contient toutes les balles de type « Balle » présentes dans le jeu.
-calculerCollision : booléen qui indique si les collisions sont calculées ou non (il faut aussi qu'il n'y ai que 2 balles dans le jeu).
-chronometre : MText du chronomètre en haut à gauche.
-croixDeVisee : booléen qui indique si la croix de visée est dessinée ou non.

-explosion : liste de dictionnaire contenant les propriétés des explosions présentes dans le jeu (avec dans les dictionnaires les attributs « pos » pour la position et « debut » pour le temps (time_ns()) de début de l'explosion).
-explosionDuree : durée maximale d'une explosion, basiquement à 0,2 secondes.
-explosionTexture : texture d'une explosion.
-fini : booléen qui indique si le jeu est fini.
-munitionsTirees : liste de dictionnaire contenant les propriétés des munitions tirées présentes dans le jeu (avec dans les dictionnaires les attributs « pos » pour la position, « profondeur » pour la profondeur sur l'axe Z et « vitesse » pour al vitesse d'éloignement).
-nbBalleDetruite : nombre de balles détruites.
-nbTir : nombre de tir total
-nbTirReussies : nombre de tirs réussies
-tempsDuDernierTir : moment (time_ns()) du dernier tir .
-texteRechargement : MText du texte de rechargement sur l'arme.
-textePub : MText du texte de publicité en bas à droite
-timecodeDebut : moment (time_ns()) du début du jeu .
-timecodeFin : moment (time_ns()) de la fin du jeu .
-_isClicked : booléen qui indique si le moteur de jeu est clické ou non.
-_positionSourisActuel : position de la souris sur la zone de jeu.

Après la définition de ces 3 classes, on défini toutes les parties graphiques de l'interface graphique de la ligne 605 à 763 en faisait la page d'accueil, la page d'option de jeu et la page de fin de jeu. On configure ensuite chaque partie graphique une par une. Ensuite, on rentre dans la boucle principale du programme.

La boucle principal :

La boucle principal permet l'exécution du programme. Elle contient 4 boucles différents qui équivalent aux 4 parties du jeu : accueil, option, jeu et fin. Chaque boucle contient mapp.frameEvent() pour gérer les évènements dans MLib, mapp.frameGraphics() pour faire le rendu graphique et pygame.display.update() pour mettre l'interface Pygame à jour.

Écran d'accueil :

La première boucle permet d'afficher l'écran d'accueil. Avant son exécution, on se débrouille pour afficher la page d'accueil et remettre à 0 l'image de « comment jouer » à l'image basique. Elle permet avec 2 boutons d'accéder au jeu ou à l'écran « comment jouer ». Elle s'occupe aussi de l'écran « comment jouer » qui est simplement une image qu'on peut changer avec 2 boutons. Après, la page d'accueil est caché.

Écran d'option :

La deuxième boucle permet d'afficher l'écran d'option. Avant son exécution, on se débrouille pour afficher la page d'option. On définit l'arme de base utilisée et la variable pour le nombre de balle. Le programme permet dans l'entrée du nombre de balle de rentrer que des chiffres. De plus, les boutons des armes et fait en « checkBox » pour choisir une arme. Dés que tout ça est choisis,

l’écran d’option est caché. Si le bouton retour est cliqué, la syntaxe « continue » est utilisée. Si « jouer » est cliqué, on passe à l’écran de jeu

Jeu :

La troisième boucle permet d’afficher l’écran de jeu. Avant son exécution, le moteur de jeu est défini avec les données saisis dans l’écran d’option. Comme tout est déjà défini dans Game, on cherche juste à savoir si le jeu est fini depuis 1 seconde pour arrêter la boucle, et détruire le moteur de jeu.

Écran de fin :

La quatrième boucle permet d’afficher l’écran de fin. Avant son exécution, on met tous les textes avec les statiques nécessaires présentes dans Game. La boucle permet de détecter le clique sur les 2 boutons, un qui recommence la boucle principale (« rejouer »), et l’autre qui arrête le programme (« quitter »).

Difficultés rencontrés

La seule difficulté rencontrée était la physique des balles et particulièrement la collision. J’ai essayé beaucoup de formules que j’ai cherché un peu partout sur Internet, allant de Wikipédia à ChatGPT. Or aucune ne marche. Seul la formule que vous nous avez donné marchait correctement. Cependant, avant celle-la, étant très souvent chez moi à cause de la maladie, j’ai passé un temps très important a trouver un moyen de gérer les collisions, ne sachant pas qu’elles ne s’appliquaient que sur 2 balles. Une autre difficulté plus approximatif est le fait que tout mon temps libre m’a permis d’avoir plein d’idées très aléatoire et n’allant pas ensemble, qui m’ont fait perdre beaucoup de temps entre leur implémentation et leur retrait si elles n’allaient pas avec le projet. Une meilleur organisation m’aurai fait gagne beaucoup de temps.

Améliorations possibles

Une quantité extrême d’améliorations sont possibles. Autant d’améliorations sont possibles que de jeux différents, soit une infinité. Par exemple, il est possible d’ajouter plein d’arme, un menu de meilleur qualité, des mécanique de jeux (lancé de grenade, drone, missile guidé...). Il est aussi possible d’ajouter un axe Z pour plus de possibilité, un meilleur affichage des balles. Pour cela, il faudrait plus de temps et une meilleur organisation. Les possibilités sont infinies.