Chapitre 7
Créer des jeux vidéo avec Pygame

1. Introduction

1.1 Contexte

Dans la plupart des langages de programmation, des fonctions permettent d'interagir avec le clavier, voire la souris. Mais dès qu'il s'agit de produire un son, d'effectuer des affichages à 30 images par seconde ou d'utiliser le joystick, nous sommes face à un grand vide. La cause profonde vient du contexte historique. En effet, ces besoins correspondent essentiellement à ceux de l'industrie du jeu vidéo. Par conséquent, des librairies spécialisées (OpenGl, DirectX) et des environnements dédiés (Unity3D, Unreal Engine…) proposent des packages spécifiques pour gérer la 3D, la 2D, les animations, les effets spéciaux, les interactions avec les joysticks, la gestion des Kinect ou des casques de réalité virtuelle…

En fait, pendant longtemps, l'offre s'est centrée autour de la librairie SDL (*Simple DirectMedia Library*) permettant un contrôle de l'audio, du clavier, de la souris, du joystick et de la carte 3D. Effectivement, proposée comme une surcouche de DirectX et d'OpenGL, cette librairie pouvait penser offrir une forme de simplicité, comparée à ces deux mastodontes destinés aux développeurs de haut niveau.

Malheureusement, SDL étant proposée en langage C, sans environnement de développement dédié, son utilisation avec les fameux pointeurs du langage C a dérouté plus d'un utilisateur. Développer un jeu très simple relevait du défi personnel dans ce contexte et mieux valait ne pas s'y aventurer sans une aide extérieure.

Heureusement, la librairie SDL a été portée sous Python, donnant naissance à Pygame. Ainsi, nous avons accès quasiment à la même puissance graphique que les librairies DirectX et OpenGL, sans toutefois avoir accès à toutes les options de dernière génération, mais nous n'en sommes pas à ce niveau-là pour l'instant ! Nous contournons cependant cette fois l'utilisation des pointeurs et la gestion de la mémoire bas niveau propre au langage C et cela fait beaucoup de travail difficile, chronophage et d'intérêt faible qui disparaît. Nous pouvons donc avec le temps disponible nous consacrer à l'essentiel : la logique des interactions et les affichages.

Restons réalistes, nous n'allons pas programmer la prochaine scène d'intro de FarCry ou de Fallout. Les grosses productions de jeu 3D, les fameuses productions Triple-A à gros budgets, nécessitent des dizaines et des dizaines de graphistes 2D et 3D et c'est eux qui rendent les jeux si beaux. Les game designers et les scénaristes sont chargés de les rendre intéressants et ludiques. Quant aux programmeurs, ils ont la lourde tâche de faire en sorte que tout cela marche !

Les productions de jeux vidéo sur téléphone portable sont déjà plus modestes : une dizaine de personnes, parfois seulement trois. Et on arrive encore à trouver des développeurs solo souvent à l'origine de la création de jeu d'auteur. Nous parlons du monde des *indie games*, c'est-à-dire des jeux crées par des équipes indépendantes composées d'un ou de quelques développeurs, avec des moyens bien moindres comparés à ceux des studios Triple-A.

À notre niveau, nous pouvons créer des jeux 2D. Produire un jeu 3D est plus lourd sous tous les aspects : technique plus difficile, plus de lignes de code à mettre en place, temps de production allongé, plus de monde dans l'équipe car il faudra au moins un graphiste 3D et un animateur 3D…

Avec le temps que nous pouvons y consacrer, 10 minutes à 3 heures, nous pouvons aborder et reproduire certains classiques de la 2D, c'est déjà très ambitieux pour un programmeur débutant ! Avec Pygame et ce livre, cela devient possible !

Nous avons fait en sorte que les premiers jeux n'utilisent pas de fonctions. Ainsi, en connaissant les bases de Python et en ayant fait quelques exercices, vous pouvez commencer à programmer des jeux vidéo !!

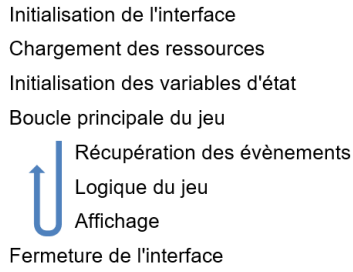
1.2 La structure d'un jeu interactif

Structurer un programme n'est pas chose aisée. Cela est bien plus dur que de programmer ou de maîtriser un langage, car cela requiert des années d'expérience. Certains vous diront que c'est facile et "qu'il n'y a qu'à faire des fonctions". Ce n'est pas si simple et ceux qui ont le jugement facile sont souvent restés cantonnés à des programmes ne dépassant pas 100 lignes.

Nous vous conseillons d'avoir une méthode et d'essayer de vous y tenir. En effet, sur un petit jeu, le programme peut vite passer de 100 lignes à 1 000 lignes, voire 10000 lignes s'il y a plusieurs niveaux, un écran d'accueil ou des ennemis différents. On ne gère pas un programme de 1 000 lignes comme on gère un programme de 100 lignes : il faut des repères, sinon le code va vite se transformer en véritable jungle. Mais à la limite, cela arrivera et ce n'est pas grave, car il faut toujours une première fois pour se rendre compte après qu'il y avait plus simple ou plus élégant comme approche. C'est cela l'expérience. N'hésitez pas à faire comme vous pouvez, car ceux qui essaient de faire parfaitement dès le départ n'arrivent pas à démarrer ou à terminer (ce qui, dans tous les cas, veut dire que le jeu est inachevé). La programmation reste un art, ce n'est pas une méthode exacte. Devenir plus expérimenté en programmation passe par l'apprentissage, les essais et les échecs.

Nous vous présentons une structuration que vous allez réutiliser dans l'ensemble des jeux de ce chapitre. Cette structuration reste un classique du monde des jeux et elle facilite grandement la vie des développeurs.

Voici les différentes étapes de traitement :



Étape 1 : Initialisation de l'interface

Cette étape crée la zone d'affichage, soit en plein écran, soit dans une fenêtre. On peut ici fixer la taille de la fenêtre, son titre ou encore l'icône qui va s'afficher dans le bandeau de la fenêtre et dans la barre des programmes actifs.

Étape 2 : Chargement des ressources

Un jeu, c'est souvent des graphismes et des sons. À ce niveau, on charge depuis le disque dur ces différentes ressources, appelées *assets* en anglais. Sur un jeu 3D, le chargement d'un niveau peut prendre plusieurs secondes, car il y a énormément de ressources graphiques.

Étape 3 : Initialisation des variables d'état

Pour expliquer ce qu'est un état, prenons l'exemple d'un aventurier. Il parcourt un donjon sombre et dangereux. Mais pour que cet aventurier soit réaliste, il a des caractéristiques : un nombre de points de vie, un niveau de fatigue, une vitesse maximale de course, un arc et un certain nombre de flèches. Toutes ces informations sont stockées dans des variables. Les différentes valeurs de ces variables décrivent l'état du personnage. À zéro point de vie, il est plutôt mort. Avec un niveau de fatigue de 90 %, il est épuisé et il doit dormir, car sinon sa précision au tir devient catastrophique. S'il n'a plus de flèches, il faut d'urgence qu'il trouve une autre arme… Ces états évoluent au cours du jeu en fonction des actions du joueur. Mais avant que le jeu commence, ils doivent être initialisés. Par exemple : `PV = 100`, `NbFleches = 8`, `Fatigue = 0`, `Poche = Vide`.

Étape 4 : Boucle principale du jeu

Cette boucle prend la forme d'une boucle `While` avec une condition du type "tant que le jeu n'est pas terminé". À la fin de chaque itération, on effectue le réaffichage complet de l'écran de jeu. Ainsi, nous avons :

un tour de boucle = une mise à jour de l'écran de jeu

Étape 5 : Récupération des évènements

Nous sommes à l'intérieur de la boucle de jeu. Généralement, un affichage vient de se terminer (étape 7) et nous allons faire évoluer les différents éléments/personnages. Pour cela, il faut d'abord aller vérifier si l'utilisateur a appuyé sur quelque chose pour savoir ce qu'il veut faire. Par exemple, l'appui sur la [Flèche à droite] va déplacer son joueur sur la droite. Chaque action de l'utilisateur crée un évènement et plusieurs évènements peuvent être en attente de traitement lorsque nous recommençons la boucle de jeu. Par exemple, le joueur peut appuyer sur la touche [Ctrl] avec sa main gauche et cliquez avec la souris avec sa main droite. Il y aura deux évènements générés et donc deux évènements en attente de traitement.

Étape 6 : Logique du jeu

Maintenant que nous savons où l'utilisateur a cliqué et où il a appuyé sur le clavier, nous allons agir en conséquence. Ainsi, l'appui sur la barre [Espace] fera tirer son vaisseau. Mais pas seulement ! C'est ici que la notion de logique intervient. En effet, il se peut que le vaisseau ait un nombre limité de missiles, et dans ce cas, même si l'utilisateur demande un tir, il ne se passera rien. De la même manière, l'appui sur la [Flèche à droite] déclenchera le déplacement d'un aventurier sur la droite, mais pas seulement. Il faudra faire des tests pour savoir si un mur est présent et donc empêcher son déplacement. Si un trou ou un piège se trouve à cet endroit, il faudra déclencher une chute. La logique recouvre ainsi toute la gestion des états du jeu. À la fin de cette étape, toutes les variables d'état ont été mises à jour. Par exemple, le nombre de missiles, s'il y a eu un tir, est diminué de 1. Si le personnage est tombé dans un trou, son état passe de "marche" à "chute".

Étape 7 : Affichage

Il faut à ce niveau recréer de zéro l'affichage de l'écran. Considérons que nous partons d'un écran tout blanc ou tour noir. Il faut ainsi tout redessiner. Souvent, il y a un fond, c'est ce qui sera affiché en premier. Ensuite, nous dessinons les différents personnages et le décor : les murs, les portes et les bonus à collecter. En dernier et au-dessus de tout ce qui a été affiché précédemment viendront les informations du jeu, comme le score ou la vie. Un élément affiché après un autre peut le recouvrir s'il s'affiche au même endroit à l'écran.

Les éléments s'affichent ainsi dans un certain ordre suivant la couche à laquelle ils appartiennent : le fond, le décor, les informations de jeu. Cet ordre reste immuable, on affiche d'abord les éléments du fond, puis les personnages et le décor, et enfin les informations de jeu.

Remarque

On mesure la fluidité du jeu en mesurant le nombre d'affichages par seconde effectués par le programme. C'est à cela que sert l'indice des FPS : (Frame Per Second). Au-dessus de 30 FPS, on peut considérer que le jeu est fluide, réactif et agréable à jouer. Lorsque l'on pousse au maximum la qualité des effets graphiques dans les jeux PC ou console, l'indice FPS chute et descend vers les 10 FPS. À ce niveau-là, l'affichage est saccadé, les commandes répondent avec du retard, jouer devient désagréable. L'aspect temps réel a disparu, le jeu "rame" et il devient énervant. C'est à ce moment-là que la décision de changer de machine est prise !

1.3 Séparation de la logique et de l'affichage

Dans la structuration que nous avons présentée, il y a une séparation entre l'étape de logique et l'étape d'affichage. Cela est rendu possible grâce aux variables d'état. L'étape de logique modifie les variables d'état, mais elle n'affiche rien. L'étape d'affichage tient compte des variables d'état, mais ne se préoccupe pas des évènements du joueur et ne modifie aucune variable d'état. Cette séparation est très importante.