L'état du jeu et le rendu

Dans un jeu vidéo, deux des tâches les plus essentielles de la boucle de jeu sont la **mise à jour de l'état du jeu** et le **dessin (ou rendu) du jeu** à l'écran. Ces deux étapes se produisent de manière répétée à chaque itération de la boucle pour que le jeu puisse réagir en temps réel aux actions du joueur et afficher les résultats de ces actions.

Dans ce chapitre, nous allons explorer en détail ces deux concepts, pourquoi ils sont cruciaux et comment ils sont organisés. Nous utiliserons également PyGame pour illustrer ces concepts avec des exemples concrets.

Mise à jour de l'état du jeu

La **mise à jour de l'état du jeu** correspond à la partie de la boucle où l'on modifie les éléments du jeu en fonction des événements qui se sont produits (comme les interactions du joueur ou les changements dans le monde du jeu). Cela peut inclure des actions comme déplacer un personnage, détecter des collisions, mettre à jour des scores, ou changer de niveau.

On parle ici de mettre à jour le **modèle**, les données du jeu, pas les graphismes. Tout ce qui compose votre jeu est d'abord et avant tout **représenté sous forme de données, de variables et d'objets** que vous pouvez manipuler avec du code.
Par exemple, la position de votre personnage à l'écran est enregistrée dans les variables `pos_x` et `pos_y`, sa vitesse de déplacement dans la variable `player_speed`, ses points de vie dans `life_points`, son score dans `score`...
L'entièreté de ce qui se passe dans le jeu est enregistré dans des variables.

Pourquoi la mise à jour est-elle importante ?

Sans cette étape, votre jeu ne pourrait pas évoluer. Par exemple, si vous appuyez sur la touche fléchée pour déplacer un personnage, la mise à jour est ce qui prend en compte cette action et change la position du personnage. Cela permet de rendre le jeu interactif et dynamique, car l'état du jeu change en permanence pour répondre aux actions du joueur et aux événements qui surviennent.

Qu'inclut la mise à jour de l'état du jeu ?

1. **Déplacement des objets** : Modifier la position des objets sur la base des entrées du joueur ou des événements du jeu. Cela peut inclure le déplacement du personnage principal, des ennemis, des projectiles, etc.
2. **Gestion des collisions** : Vérifier si des objets dans le jeu entrent en contact les uns avec les autres, comme un personnage avec un ennemi ou un projectile avec un mur.
3. **Mise à jour des animations** : Changer les images (sprites) affichées pour donner l'illusion du mouvement ou d'une animation fluide.
4. **Évolution des statistiques du jeu** : Modifier des variables telles que les points de vie, les scores, ou le niveau de difficulté.
5. **Génération de nouveaux objets ou suppression d'objets** : Ajouter ou retirer des éléments du jeu, comme des ennemis ou des bonus, en fonction des conditions définies (ex : lorsqu'un ennemi est éliminé, il est retiré de l'écran).

Exemple simple de mise à jour dans PyGame

Imaginons que nous avons un joueur représenté par une image (sprite) et que nous voulons déplacer ce joueur vers la gauche ou la droite en fonction des touches fléchées. Voici un exemple de code pour la mise à jour de l'état du joueur.

```
1  import pygame
2
3  # Initialisation de PyGame
4  pygame.init()
```

```python

# Création de la fenêtre de jeu
screen = pygame.display.set_mode((800, 600))
pygame.display.set_caption("Mise à jour de l'état du jeu")

# Chargement de l'image du joueur
player = pygame.image.load('player.png')
player_x = 100
player_y = 300
player_speed = 5  # Vitesse du joueur

# Boucle de jeu principale
running = True
while running:
    # Gestion des événements
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    # Récupérer les touches pressées
    keys = pygame.key.get_pressed()

    # Mise à jour de la position du joueur
    if keys[pygame.K_LEFT]:
        player_x -= player_speed
    if keys[pygame.K_RIGHT]:
        player_x += player_speed

    # Remplir l'écran avec une couleur de fond
    screen.fill((0, 0, 0))

    # Dessiner le joueur à sa nouvelle position
    screen.blit(player, (player_x, player_y))

    # Mettre à jour l'affichage
    pygame.display.flip()

    # Limiter la boucle à 60 images par seconde
    pygame.time.Clock().tick(60)

pygame.quit()
```

Explication de la mise à jour dans cet exemple

1. **Déplacement** : Ici, on utilise `pygame.key.get_pressed()` pour récupérer les touches appuyées, puis on modifie la position `player_x` en fonction des touches fléchées gauche et droite. Cela permet de mettre à jour l'état du joueur à chaque cycle de la boucle de jeu.
2. **Framerate** : Pour s'assurer que la mise à jour se produit à une vitesse constante, nous limitons la vitesse de la boucle à 60 itérations par seconde (`60 FPS`), garantissant une mise à jour fluide et régulière.

Dessiner le jeu (rendu graphique)

Une fois que l'état du jeu a été mis à jour, il faut ensuite **dessiner** tous les éléments à l'écran pour que le joueur puisse voir ce qui se passe. C'est ce qu'on appelle le **rendu** du jeu. Cette étape consiste à effacer l'écran précédent et à redessiner tous les objets (personnages, décors, interface) à leur position actuelle.

Pourquoi le dessin est-il essentiel ?

Le dessin permet de **visualiser** ce qui se passe dans le jeu. Si un personnage se déplace, il faut que cette nouvelle position soit représentée à l'écran. Sinon, le joueur n'aurait aucun retour visuel sur ses actions. Le rendu graphique se produit à chaque itération de la boucle de jeu pour donner l'illusion d'un mouvement fluide et d'un monde interactif.

Qu'inclut le dessin du jeu ?

1. **Effacement de l'écran** : Avant de dessiner une nouvelle scène, il est important de nettoyer l'écran pour éviter que les anciennes images ne se superposent. C'est souvent fait en remplissant l'écran avec une couleur de fond.
2. **Dessin des objets du jeu** : Après avoir effacé l'écran, on dessine tous les objets (joueur, ennemis, projectiles, etc.) à leur nouvelle position.
3. **Interface utilisateur (UI)** : En plus des éléments du jeu, on peut dessiner l'interface utilisateur, comme des barres de vie, des scores, ou des menus.
4. **Actualisation de l'affichage** : Une fois que tout est dessiné, il est nécessaire de demander à PyGame de mettre à jour l'affichage à l'écran pour refléter ces modifications.

Exemple simple de dessin dans PyGame

Reprenons l'exemple précédent et concentrons-nous maintenant sur le processus de dessin (ou rendu). À chaque itération de la boucle de jeu, nous remplissons d'abord l'écran avec une couleur de fond (pour éviter les "traces" du joueur précédent), puis nous dessinons le joueur à sa nouvelle position.

```
# Remplir l'écran avec une couleur de fond
screen.fill((0, 0, 0))  # Noir

# Dessiner le joueur à sa nouvelle position
screen.blit(player, (player_x, player_y))

# Mettre à jour l'affichage
pygame.display.flip()
```

Explication du dessin dans cet exemple

1. **Effacement de l'écran** : L'appel à `screen.fill((0, 0, 0))` remplit toute la fenêtre avec la couleur noire (RVB : 0, 0, 0), effaçant ainsi l'affichage précédent.
2. **Dessiner le joueur** : Nous utilisons `screen.blit(player, (player_x, player_y))` pour dessiner l'image du joueur à sa nouvelle position après avoir mis à jour sa position.
3. **Actualisation de l'affichage** : L'appel à `pygame.display.flip()` est crucial. Il permet de rafraîchir l'écran et de rendre visibles les changements graphiques. Sans cela, même si vous dessinez les éléments, ils ne seront pas affichés correctement.

Relation entre la mise à jour et le dessin

La **mise à jour** et le **dessin** sont étroitement liés, mais ce sont deux étapes distinctes. La mise à jour de l'état du jeu modifie les données du jeu (position des objets, état des personnages, etc.), tandis que le dessin utilise ces données pour afficher une représentation visuelle de l'état actuel.

Voici comment ces deux étapes interagissent :

1. Le joueur appuie sur une touche pour déplacer un personnage (entrée utilisateur).
2. La position du personnage est mise à jour dans l'étape de **mise à jour**.
3. Le nouveau placement du personnage est ensuite utilisé dans l'étape de **dessin** pour afficher le personnage à la nouvelle position.
4. Le cycle recommence à chaque itération de la boucle de jeu.

Dans les jeux modernes, ces étapes doivent être rapides et optimisées pour que le joueur ne remarque aucun délai, assurant ainsi une expérience fluide.

Optimisations liées à la mise à jour et au dessin

Minimiser les calculs inutiles

Un bon jeu doit être réactif, mais il ne doit pas non plus consommer trop de ressources. Pour cela, vous pouvez :

- **Optimiser les calculs** : Ne mettez à jour que les objets qui changent, plutôt que de recalculer l'état de chaque objet à chaque cycle de la boucle.
- **Rendre uniquement ce qui est nécessaire** : Si certains objets ne changent pas (comme un fond de scène statique), vous pouvez éviter de les redessiner à chaque itération de la boucle.

Gestion efficace des sprites

Dans des jeux avec

de nombreux éléments graphiques (sprites), utiliser des **Groupes de sprites** dans PyGame peut faciliter la gestion de l'affichage et améliorer les performances. Cela permet de regrouper des sprites similaires et d'effectuer des opérations collectivement (comme les dessiner tous ensemble ou vérifier les collisions).

Conclusion

La mise à jour de l'état du jeu et le dessin sont deux concepts clés dans la création de jeux vidéo. Ils sont à la base de tout système de jeu interactif, permettant aux joueurs de voir les effets de leurs actions et de s'immerger dans l'expérience. En maîtrisant ces deux concepts et en les organisant correctement dans la boucle de jeu, vous serez en mesure de créer des jeux dynamiques, réactifs et visuellement attractifs.